EXTENSIBLE OPTICAL MUSIC RECOGNITION

A THESIS

SUBMITTED IN PARTIAL FULFILMENT

OF THE REQUIREMENTS FOR THE DEGREE

OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

AT THE

UNIVERSITY OF CANTERBURY

by

David Bainbridge

University of Canterbury

1997

This thesis is dedicated to my family.

Acknowledgements

First I thank my supervisor, Tim Bell, for his guidance over the years. Without his support and advice there would be no thesis. Also, criticism on the final draft of the thesis by Tanja Mitrovic and proofreading by Paul Kennett were greatly appreciated.

The technical support given by the Department of Computer Science at Canterbury was invaluable—thanks Tony, Joff, Pete, Blair, and Gill. In the latter part of my Ph.D. at the University of Waikato, technical assistance by Brent and Roger at key times helped immensely.

The post-graduates I met (both past and present) made the whole thing worth while. There are too many names to list them all, but I especially thank Mike, Hugh, Nigel, and Lindsay for their late night entertainment.

Lastly I thank my fiancée and soon to be wife, Annette, for enduring the writing process with me.

This research was supported by a Commonwealth scholarship.

DAVID BAINBRIDGE

University of Canterbury

January 1997

Abstract

The aim of Optical Music Recognition (OMR) is to convert optically scanned pages of music into a versatile machine-readable format. Existing work has achieved this aim for restricted sets of music notation. Here we investigate the design of an extensible OMR system.

Music notation is characterised by intricate features which prove too complex for current computer systems to recognise in a single step. A common methodology in OMR systems is to detect simple primitive shapes which are then assembled into the intricate musical features. However, developing a system capable of processing an extensible set of notation is problematic because there is no limit to the musical shapes that can occur. This thesis deals with the issue by combining a specially designed programming language for primitive detection, a user-configurable knowledge-base for primitive assembly, and an object oriented interface for musical semantics. In doing so, the design is capable of processing not only an extensible set of shapes within one notation, but a variety of notations, such as common music notation, plainsong notation, and tablature.

The specially designed programming language eliminates the need for repetitive descriptions, and consequently the code is concise. Grammar rules in the knowledge-base provide a flexible medium in which the valid taxonomy of musical features can be expressed. Finally, the object oriented interface provides a mechanism that can be tailored to encode the semantics of a specific musical notation.

Within this framework, the thesis investigates six important steps in the OMR process—staff detection, musical object location, image enhancement, primitive detection, primitive assembly, and musical semantics. Existing work is refined and new algorithms are developed where appropriate. The thesis concludes by comparing the performance of two OMR configurations aimed at reliable matching. Both take approximately 10 minutes to process an A4 page of music using a Digital Celebris GL 5133, with an overall accuracy rate that exceeds 96%.

Contents

Chapter 1

Introduction

Imagine a computer system that could "read" printed music: you could listen to a piece of written music without any training in musical notation; a clarinetist could scan a tune and have it transposed automatically; a soloist could have the computer play an accompaniment for rehearsal; a music editor could make corrections to an old edition using a music notation program; or a publisher could reduce an orchestrated work to a piano part with little fuss, and convert the piece to Braille with almost no extra work. This is the aim of Optical Music Recognition (OMR): to convert optically scanned pages of music into a versatile machine-readable format.

Optical music recognition addresses the problem of *musical data acquisition*, the key impediment to the success of the above examples. It is not, however, the only data entry method for music. Currently, the most common method for music data entry uses a combination of synthesiser keyboard entry and computer keyboard entry. The musical keyboard is used to enter the notes by playing each voice in isolation, typically in time with a metronome; the computer keyboard and mouse are then used to correct any mistakes and to add other notation such as lyrics, slurs, and dynamics. Music data entry in this form demands a high level of skill from the keyboard player and adding the remaining notation is time-consuming.

Even if current commercial software could be refined to record information such as dynamics from the synthesiser keyboard, the computer keyboard stage will always be required, since there are many more features to printed music than the way it is finally played—for instance, clefs, key signatures, time signatures, titles, and lyrics. Also, decid-

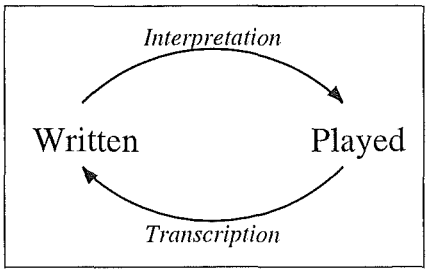


Figure 1.1: Forms of music representation.

ing on line breaks, page breaks, even the spacing and grouping of notes within a bar, is a skilled task, and is generally seen as a craft rather than some process that can be defined by a set of rules and automated [Ros70, Rea74, Heu87].

We are not arguing, however, that OMR should replace musical keyboard entry. In the particular circumstance where the music is already available in printed form, OMR can substantially accelerate the process of musical data acquisition. OMR, therefore, is an attractive supplement. Not only does the technique reduce the chance of human error made in the transcription, but it can also capture much of the "extra" information that the former method requires the user to laboriously add after the notes have been played.

The most likely scenario is one where OMR is used to process the majority of symbols on the page, followed by an editing stage using a standard music editor, where the musical and computer keyboards are used to correct mistakes and omissions. This has the added benefit of greatly reducing the musical keyboard skills required. Given the vast body of printed music, OMR could radically reform computer applications in music.

To help clarify the limits in musical data acquisition by computer, let us now consider the larger context of music representation. Figure 1.1 shows the relationship between written and played music. Written music is converted to played music by *interpretation*, and played music is converted to written music by *transcription*. Both are non-trivial operations. To give a performance, a soloist may study the written work for months, deciding how the piece is to be played, before finally presenting their interpretation. The performer will draw upon a range of information, such as knowledge about the composer, and the technical limits of the instrument, as well as considering the mood and feeling of the work. To transcribe an imagined performance into a written form, a composer reverses this process, deciding what notation would indicate the desired effect. A similar process

is undergone when a recording that has never before existed in written form—such as a jazz improvisation, or a folk tune—is transcribed.

Only limited success has been achieved by computer techniques that imitate these human processes. A computer application that requires the translation from written music to an audio equivalent or *vice versa* will inevitably suffer from a loss of information. Such a situation occurs in synthesiser keyboard data entry, and explains the strong reliance on post editing. Equally, an OMR system that is used for audio playback results in a mechanical sounding rendition. Better results are achievable if the computer music application stays within one medium: a MIDI keyboard can be used to great affect in sequencing work; and an OMR system is ideal for editing the written page.

1.1 Properties of music

In principle Optical Music Recognition (OMR) is the extension of Optical Character Recognition (OCR) applied to music; however, the problems to be faced are substantially different. Below, examples are provided that demonstrate the complexities involved. First we consider the problems posed by *individual* musical features, then we broaden our view and look at the relationships that exist *between* musical features.

Due to geographic and cultural reasons more than one music notation has developed over the years. Historically known as Western Music Notation [Gro60], Common Music Notation (CMN) has become the notation most frequently used around the world, and is therefore the principal notation discussed in this thesis. Although the discussion of musical properties below is set in the context of CMN, much of the discussion applies to other notations.

A musical feature is typically more intricate than a text character, consisting of a variable numbers of components that can change size, orientation, and position. Often the difference between two musical features is a subtle alteration in the component construction—as in the case of notes: minim, crotchet, quaver and so on. There are also many musical features that include multiple, disjoint components—such as key signatures, and the bass clef.

Further information is conveyed through elaborate two-dimensional relationships that exist between musical features. For example, the pitch of a note is conveyed through:

the *superimposition* of its note head on the staff; the effect of the closest preceding clef and key signature; and the potential presence of an accidental at the same pitch earlier in the bar. In addition to this, a line of notation in isolation may be *ambiguous*, requiring the study of a wider context to resolve the matter.

1.1.1 Individual musical features

The graphical properties of musical objects are significantly different to those of printed text. This point is illustrated in Figure 1.2. Text utilises the vertical dimension of the page in a simplistic way, spacing subsequent lines of text in an orderly fashion. Each line itself conforms to a base-line stretching horizontally across the page. Music extends the use of the vertical dimension further. Within a line of music, the y-axis is also used to convey pitch; the same shape, therefore, can be translated to different vertical positions. The translation example in Figure 1.2 shows three crotchet notes that are graphically identical, but drawn at different positions indicating different pitches. The issue of translation is additionally complicated by chords, since individual note heads are no longer constrained to be at the end of stems. Examples of this selective translation are also shown in Figure 1.2. In principle there are an infinite number of variations in the placement of note heads in a chord.

Other requirements of music lead to the same musical event having more than one graphical representation. In some circumstances, the difference is a simple alteration such as stretching. This can occur horizontally, as in the slur and beamed note examples in Figure 1.2, the main cause of this being the vertical alignment of notes between staves that are played at the same time. Shapes may also be stretched vertically, a good example being a beamed note where the note stems have been either lengthened or shortened to connect with the beam. More complex alterations include rotation, which is dependent on where the note is placed on a staff, or whether the staff is carrying two voices at that point; and shearing, a consequence of using vertical displacement to indicate pitch.

Another important difference between text and music is that the graphical appearance of each text item is intentionally different. In music, however, shapes are sometimes graphically similar, where the minor differences convey extra information. In Figure 1.2 the example for similarity shows a progression of notes where each successive shape is slightly different to the last, yet each change doubles the duration of the note. A simi-

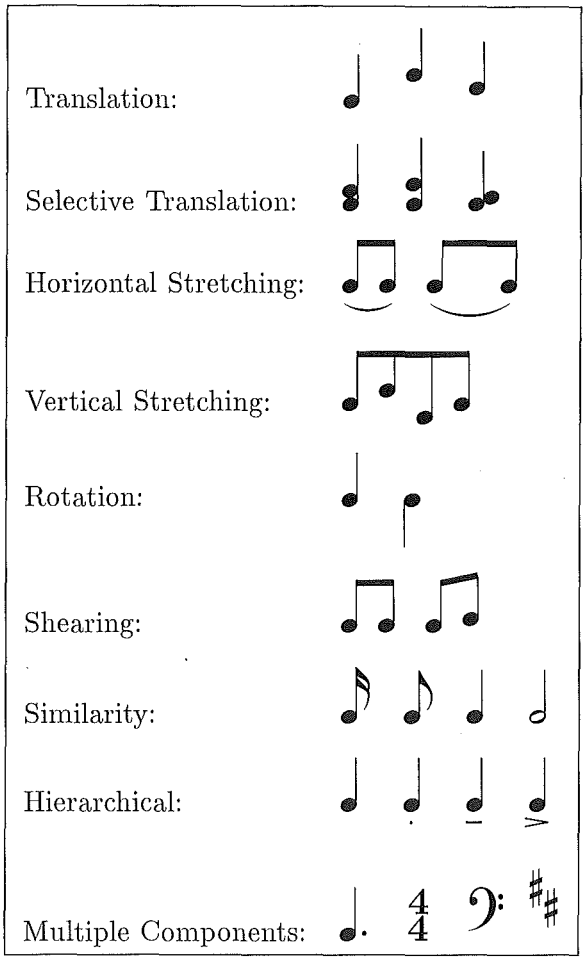


Figure 1.2: Properties of music not found in printed text.

lar situation also occurs with hierarchical symbols in music. By hierarchical we mean optional symbols that can occur in addition to the main musical feature, to convey extra information. In Figure 1.2, the hierarchical example shows the same basic crotchet note altered to become a staccato note, a stressed note, and an accented note, by the addition of an extra component below the note head. This additionally serves to illustrate the frequent use, in music, of multiple components. Also included in Figure 1.2 are more varied examples of shapes that use multiple components. This contrasts with Western text, where multiple components are relatively rare, and typically consist of a main shape with a supporting smaller mark. Most common is the use of a dot, as in 'i', 'j', ':', ';', '?', and '!', with some European languages augmented by diacritical marks, such as 'é', 'è', and 'ö'. Such exceptions are normally handled by OCR systems using ad hoc methods.

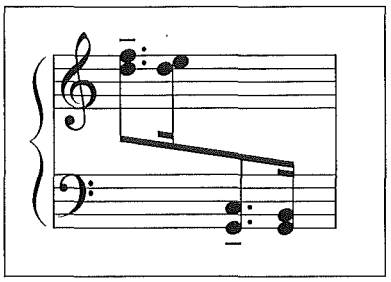


Figure 1.3: A musical feature can include many variations of representation.

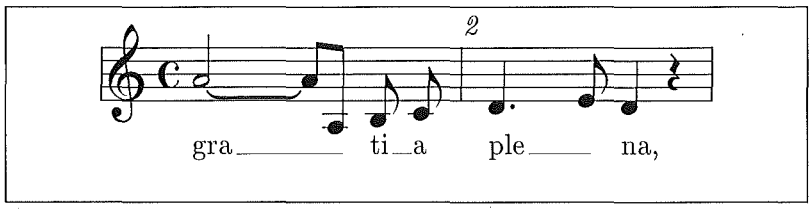


Figure 1.4: Music relies on many complex two-dimensional spatial relationships.

To compound these difficulties for an OMR system, these transformations do not occur in isolation. One musical shape may vary all these properties simultaneously. Figure 1.3 shows one example where the construction of the note includes stretching, shearing, rotation, selective translation, and hierarchical symbols.

1.1.2 The layout of musical features

The differences between music notation and text go further than this variance in the atomic characters. Text is predominately one-dimensional in layout, whereas music makes full use of the two-dimensional space. For example, a clef at the start of a line of music affects subsequent notes on that staff until another clef is encountered, or the end of the staff is reached; and the syllables to lyrics written beneath a staff are associated with particular notes. These points are illustrated in Figure 1.4.

Complex two-dimensional relationships are characteristic of Document Image Analysis (DIA) problems, an active area of research [BBY92]. Figure 1.5 shows some examples of document types that have been processed successfully by computer. To recognise such documents the general strategy is to first segment the image, then to correctly form the larger objects that are present in the image, from the segmented components. This

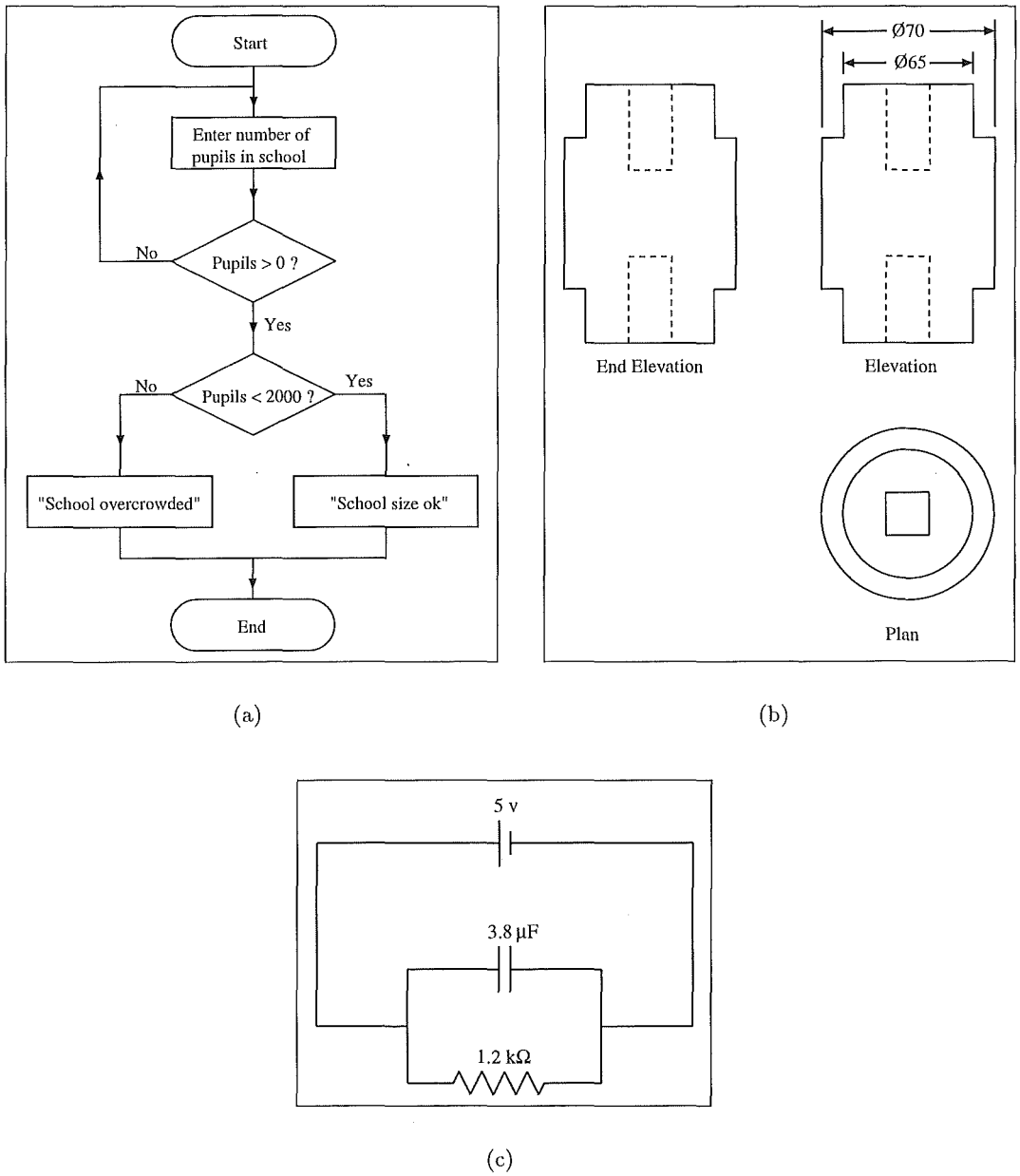
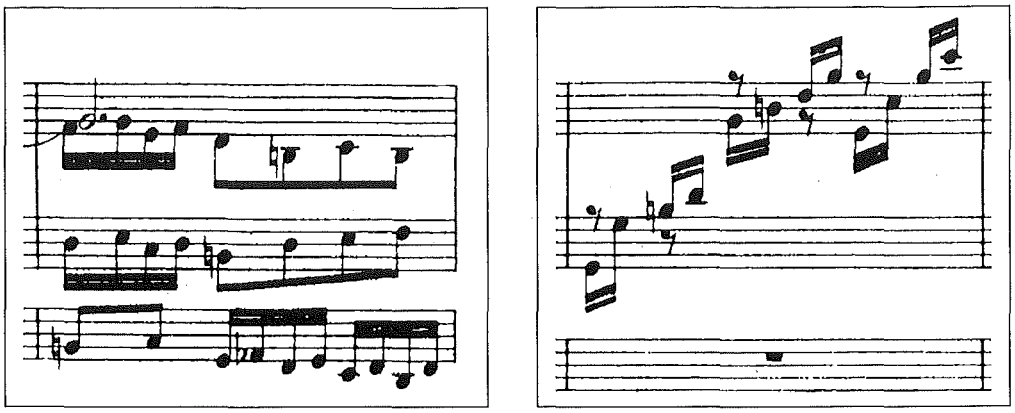


Figure 1.5: Examples of computer processed documents (a) flowchart (b) technical drawing (c) circuit diagram.



(a) (b)

Figure 1.6: Music notation is sometimes written ambiguously to improve layout (a) the minim note is played at the start of the bar, despite being positioned further into the bar (b) the bar is played as an arpeggio using alternate left and right hands, despite being drawn on staves which normally indicate separate left and right hands.

is accomplished by using some computer encoded representation of knowledge about the notation that a person skilled in the notation takes for granted once learnt. DIA forms a rich source of techniques for OMR.

Ambiguity

A consequence of music relying on two-dimensional relationships is the issue of order. When considering a single voice of music, there is generally a unique and understood order in which the music must be read, otherwise the music would be ambiguous. However, it is sometimes *written* in an ambiguous manner. In such situations, the breach of convention is normally a sacrifice intended to simplify layout, and an experienced musician can still determine the meaning of the music and correctly pick the order—typically because the alternative interpretations would be nonsense. For example, in Figure 1.6a the dotted minim on the top staff is technically drawn too far to the right, forming a separate note event, which results in a bar lasting for six beats. This of course conflicts with other information on the page, such as the visual spacing of the notes, the stem directions, and the duration of the other two bars, which are both three beats. A musician knows to play the dotted minim at the start of the bar.

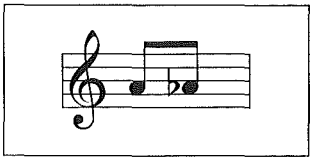


Figure 1.7: A seemingly simple piece of music relies on two-dimensional layouts and relationships that are complex and intricate for a computer to process.

Another example of ambiguity is shown in Figure 1.6b. After study, it becomes apparent that the notation shows both the left and right hands starting on the second staff, alternatively playing notes in an ascending arpeggio, with the notation crossing over to the first staff part way through. According to the strict rules of CMN though, the notes in both the first and second staves should start simultaneously, with the right-hand playing the first staff, and the left-hand playing the second staff.

Just because a correct order exists, this does not mean it will be easy for a computer to determine. For example, in Figure 1.7, consider which of the two musical shapes should be processed first: the beamed note or the accidental? Certainly the beamed note appears first in reading from left to right, however if this regime was strictly adhered to then the second note head would be processed (and possibly played) before the consequences of the flat are realised. Alternatively, the accidental finishes before the end of the beamed note, so perhaps the accidental should be processed before the two notes, which seems equally inappropriate.

1.1.3 Superimposed musical features

A unique aspect of music that does not occur even in the DIA examples shown in Figure 1.7, is the *superimposing* of shapes. In music notation, notes are intentionally superimposed on staves to convey pitch, and slurs, ties and crescendo/diminuendo "hairpins" sometimes cross over bar lines, making it difficult for computer algorithms to distinguish musical features. Similar problems of occlusion occur in other fields, such as robot vision [CPW93] and computer automated cell counting in Biology [Rus92]. In robot vision, an object at one distance can be partially hidden by an object closer to the robot's view point. In cell counting, two cells on the microscope slide may overlap, partially obscuring each other. Researchers report that superimposition is a severe problem in image pro-

cessing, and one that proves difficult to compensate for. A great deal of the complexity in OMR systems is due to superimposition.

1.1.4 Music notation includes text

So far, the discussion has concentrated on the unique musical shapes found on the staff. Music, though, includes many other shapes off the staff, such as bar numbers, dynamics, fingering information, and other text and text-like characters. Thus, OMR is really a *superset* of OCR.

To fully process the text in music, it is insufficient to convert each letter to its ASCII[1] form, and then group words and lines together. The position of the text on the page is important, and must be used to associate its meaning with the correct part of the music. To complicate things further, it is conventional for dynamic markings to be written in Italian and abbreviated (***f***, ***mf***, ***p***, and so on), and other languages are common, even if the music is intended for English speaking musicians.

1.2 Background

The first published OMR work was carried out at Massachusetts Institute of Technology in the mid 1960's by Pruslin [Pru66, Kas72]. Pruslin's system recognised a subset of CMN, primarily musical notes. It lacked features such as clefs, time signatures, and rests, but permitted complex stem-sharing chords. Prerau [Pre70, Pre71, Kas72, Pre75], originally intended to extend Pruslin's work, but unfortunately he discovered that Pruslin's staff-removal (used to tackle the issue of superimposing) distorted most musical features other than note clusters and beams. He therefore had to develop an alternative approach.

Prerau observed that, unlike text characters, a musical feature could usually be identified by the dimensions of its bounding box. However, as the same musical feature could appear larger or smaller in a different piece of music, the idea could not be generalised directly. To make his observation independent of absolute size, the dimensions of the box were normalised with respect to the height of the staff.

A survey of different styles of music was undertaken and a database built. The heuristic classification algorithm developed used the database to quickly reduce the num-

[1] ASCII is an abbreviation for American Standard Code for Information Interchange.

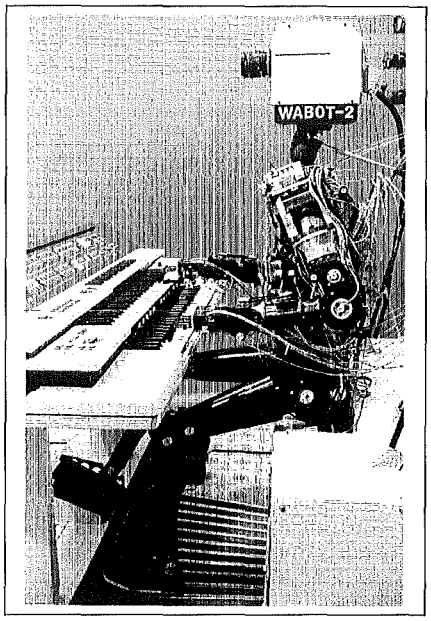


Figure 1.8: The Wabot-2.

ber of possible musical features an unknown shape might be: often the database would return a single match, and thus the object was classified. In situations where the specified regions for musical features in the database overlapped, additional checks were designed to distinguish the shapes.

In contrast to Pruslin's work, Prerau's system recognised clefs, rests, certain time signatures, and accidentals. However, it could not handle chords. Many systems following Prerau's work have made significant use of his observation about musical features having distinctive bounding boxes, leading to efficient musical feature classification algorithms.

Early work was limited by technology, and merely acquiring image data was a significant part of these projects. Prerau, for example, used a flying spot scanner to achieve a scan resolution of approximately 225 dots per inch (dpi), whereas today 300 dpi, 400 dpi, and 600 dpi scanners are common and more reliable.

It was not until the early 1980's that the next significant development occurred. In a major undertaking by Waseda University (Japan), a robotic organist (Figure 1.8) was built [MHS+85, Roa86]. Demonstrated at the 1985 International Exposition, the robot would play the organ using its mechanical hands and feet. The robot could play an unknown piece of sheet music placed in front of its electronic eye (a video camera), or ac-

company someone singing a tune it already knew. Visitors to the exposition were invited to converse with the robot, and select a piece to sing. The robot would then keep in time with the singer, even if they varied their speed, and would automatically transpose key if the person accidentally changed the key they were singing in.

The real-time processing necessary was achieved by using dedicated hardware: 17 16-bit computers and 50 8-bit computers with fibre optic data-links. The project lasted $3\frac{1}{2}$ years, and involved four professors and some 50 students. Had commercial equipment and services not been donated, it is estimated that the project would have cost US$2,000,000 [Roa86].

The OMR component of the Wabot-2 was the first system to recognise a comprehensive set of CMN, including chords. The real-time processing requirement led to a 'brute force' approach for many OMR problems. A key component of the system was a frame-buffer that was used to store a normalised version of the page. Significant use was made of template matching, rendered practical by being realised in hardware, which processed the image stored in the frame-buffer.

More recently the cost of hardware suitable for OMR has come within the budget of many research centres. A reasonable configuration would consist of a flatbed scanner with 200-400 dpi resolution, connected to a workstation with appropriate memory and disk space[2]. The fact that no hardware item is peculiar to OMR has been beneficial to the growth of the subject. Since 1985 there have been over 20 projects, ranging from Honours to Ph.D. level and beyond.

The increased interest has substantially helped further understanding of the problems in accomplishing OMR, and the first commercial packages that recognise the core of CMN are beginning to emerge. Also, some OMR research projects have concentrated on medieval music [MM91, Car92b], and handwritten formats [RT88, BAGS92, WCAK92, YBD+92]. Work with both formats is, understandably, harder since the clarity and regularity of the graphics is considerably reduced. For example, in handwritten music a note head is more likely to be physically separated from its note stem, and will more typically consist of a short stroke rather than the neat oval found in engraved scores. Results are promising, but work on these topics lag behind the success of printed CMN in much the same way handwritten OCR research follows in the wake of printed OCR.

[2] An A4 page scanned at 300 dpi takes around 80-100 Kbytes when compressed.

1.3 The end point of OMR

At the start of this chapter, applications for OMR such as accompaniment, editing and transposition were cited. These examples illustrate that the level of musical detail required from an OMR system is dependent on its intended use.

For audio playback, it is sufficient to recognise only the pitches and durations of musical features. This is the focus of current commercial OMR systems, because this is all that is required for many simple applications and is relatively easy to achieve. Items such as the title and fingering information are superfluous, and depending on how crude the playback can be, dynamic markings are also usually ignored.

Alternatively, an OMR system for editorial enhancement must recognise all the graphical shapes on the page. In this situation, typography is the primary goal, hence it is not strictly necessary for the OMR system to understand the musical significance of the graphical shapes, only what the primitive shapes are, and where they are located.

The two examples detailed above represent extreme situations. Other applications require a mixture of *graphical recognition* and *musical understanding*. The ultimate goal of OMR work is a system capable of graphically recognising *and* musically understanding all the information present in a page of music.

1.4 Technology

Hardware that is state of the art today, will be seen as cumbersome a few years hence. For this reason, the algorithms described in this thesis are presented independently of specific hardware. However, the validation of new methods must, by necessity, be undertaken using current technology. The software written for this thesis was developed using a Sun Sparc 5 workstation, with 230 MBytes of memory, running Solaris 2.5 (a variant of Unix), and evaluated on a Digital Celebris GL using a Pentium 133 processor, with 32 MBytes of memory, running Linux (another variant of Unix).

Additional hardware required for an OMR project is an optical scanner. Current designs are: hand-held, fixed-head, flatbed, and drum.

Both the hand-held and fixed-head scanners form the low end of the market. With a hand-held scanner, the operator controls the movement of the scanner head by sweeping it across the page, so the scan is vulnerable to variations in speed and direction. A

fixed-head scanner is similar in design to a fax machine, where the scanning head remains stationary, and the paper is moved past at a steady speed. Though more reliable than a hand-held scanner, it can only process loose sheets. A flatbed scanner works on the same principle as a photocopier. The document is placed on a glass pane, and the scanner head moves to cover the area of the window. Drum based scanners form the high end of the market and are priced accordingly. Used by professional printers, the document is fixed to a drum, which is then rotated in a controlled manner past the scanner head. The accuracy of the design allows, for example, 35mm slides to be processed at very high resolution.

A flatbed scanner is the most versatile design, being able to scan pages from books—an important source for music. It is also an affordable item of equipment for a research centre. This design is therefore selected for use with this project. Despite this commitment to a specific style of scanner, the design of the system will maintain scanner independence where possible. For instance, we will not be restricted by the physical dimensions of the scanning window, choosing to break a scan into sensible pieces, where appropriate. Nor will the work carried out make assumptions about the scan resolution. With today's technology, scanning at 300 dpi makes reasonable use of memory and disk resources, and forms the main scan resolution for this work, however the algorithms described in this thesis will work for any sensible scan resolution, such as 150 dpi or 600 dpi.

1.5 Musical terminology

It is assumed that the reader is proficient at reading CMN and is therefore familiar with musical terminology. Rader provides an excellent summary for the novice [Rad96].

When describing the methods involved in the processing of musical images, it is necessary to use typographically related musical terminology. This is perhaps less familiar to the reader. Figure 1.9 shows the main terms.

1.6 Synopsis

An overview of OMR is presented in Chapter 2. In this chapter, a general framework for OMR systems is proposed, illustrating five main components: *staff line identification*, *musical object location*, *optional image enhancement*, *musical feature classification*, and *musical semantics*. This chapter also highlights the complication of fragmentation caused by

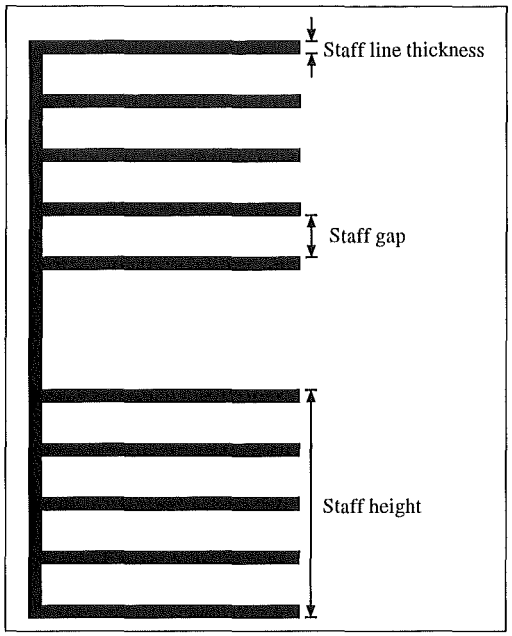


Figure 1.9: Typographically related musical terminology.

separating superimposed objects, and introduces the thesis that a practical OMR system must be able to cope with objects that are fragmented, as well as objects that are incorrectly joined. This in turn leads to a discussion on recent trends, and the main aims of the author's work.

The following chapters (Chapters 3 to 8) take the main components of an OMR system and analyse them in turn. In the case of musical feature classification, this complex task has been further subdivided into two chapters: primitive shape detection (Chapter 6) and primitive shape assembly (Chapter 7).

Each chapter follows the same structure. First a review of the existing algorithms is presented with a discussion of the strengths and weaknesses of prior work. In some situations this discussion identifies areas of deficiency in our knowledge of OMR, while in others it identifies improvements that are possible. For either situation, experimentation is required to determine the success of the work. The middle section of these chapters, therefore, describes the new algorithms and outlines tests that will either increase our understanding of the problem, or establish the relative performance of the variations. The results are presented in the last section of each chapter.

In Chapter 9 the main components are put together to form a complete OMR sys-

tem, and the issues that result by processing a scanned piece of music from start to finish are analysed and explained. To explore the issue of speed versus reliability, two different configurations for the OMR system are evaluated.

The conclusion is presented in Chapter 10. This includes a summary of the findings of this thesis, and identifies areas for further work.

Appendices at the end of the thesis are: musical excerpts from the corpus of scanned music used to test algorithms (Appendix A); and low level implementation details of a programming language specifically designed to aid the task of musical primitive shape recognition (Appendix B).

Chapter 2

An overview of OMR

Optical music recognition is a complex problem, rendered manageable through decomposition. In Figure 2.1 a general framework for OMR is proposed. The model shows how OMR can be simplified by decomposing it into four smaller tasks: *staff line identification*, *musical object location*, *musical feature classification*, and *musical semantics*. Optional *image enhancement* is possible after staff line identification and musical object location.

The proposed framework is not the only decomposition, however there is good reason to base work on this model. All of the existing OMR work fits the model [BC96], although this fact is often superficially obscured by the use of different terminology, and choices in system structuring that combine or overlap particular stages in the proposed model. In the future this framework may not prove to be the best solution for OMR, since developments in programming paradigms may prompt a different structuring; however, it is still useful since the model unifies the different strategies that exist today, allowing us to compare and contrast different OMR systems. Using the model, we can identify the strengths or weaknesses of a given OMR system, and discuss the merits of competing algorithms.

In this chapter, the four principal stages of the model are described in turn. This leads on to a discussion of recent trends, and the main aims of the author's work. For a fuller and more historic review of OMR work see Blostein and Baird [BB92], Selfridge-Field [SF94], and Bainbridge and Carter [BC96].

Figure 2.2 shows a representative excerpt of music that will be used throughout this chapter to illustrate the stages of the general OMR framework.

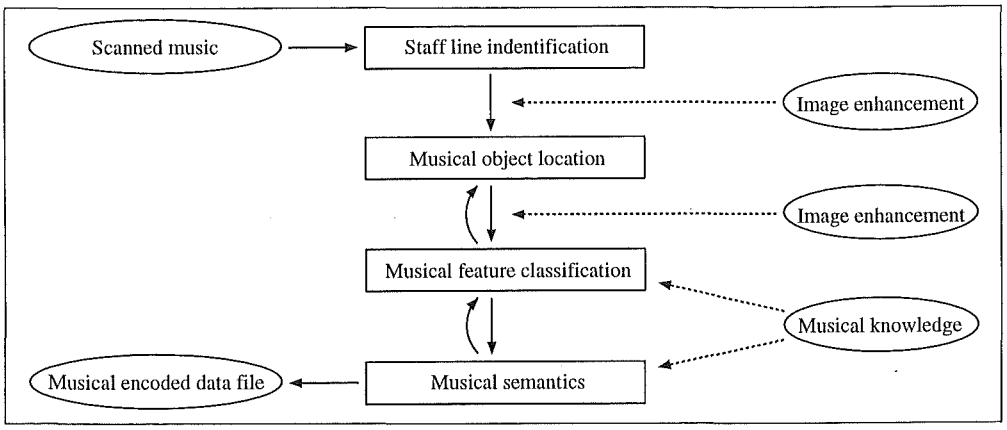


Figure 2.1: A general framework for OMR.

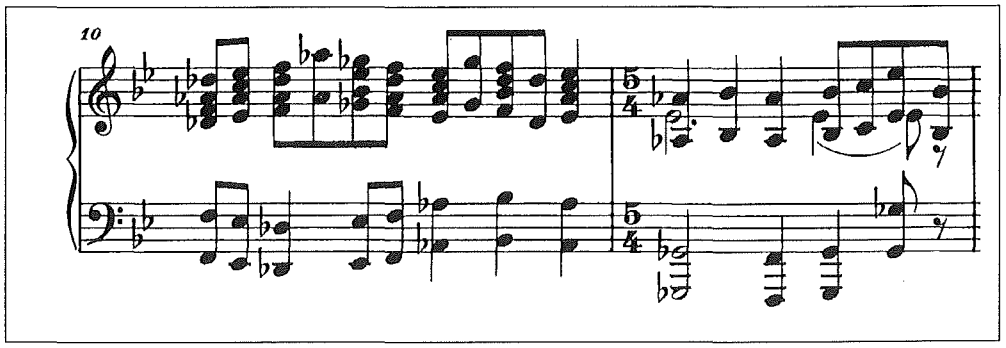


Figure 2.2: A representative excerpt of music, taken from Mussorgsky's "Pictures at an exhibition," published by Schott of Vienna, Austria (Page 2, Line 4).

2.1 The size of music notation

Before an OMR system can start recognising the shapes contained in an image, it must establish the size of the music notation being used. Staff lines are a reliable feature contained within a page, from which the staff height can be deduced, and consequently the size of the music notation. All reported OMR systems detect staff lines as the initial stage.

Since staff lines in a scanned image are not guaranteed to be level (or even straight!) detecting staff lines is a trickier task than might first be imagined. To deal with these complications, staff line detection algorithms need to make some assumptions about the image. Two common assumptions are that staff lines cover a large proportion of the image, and that (excluding noise) their thickness will be one of the smallest found.

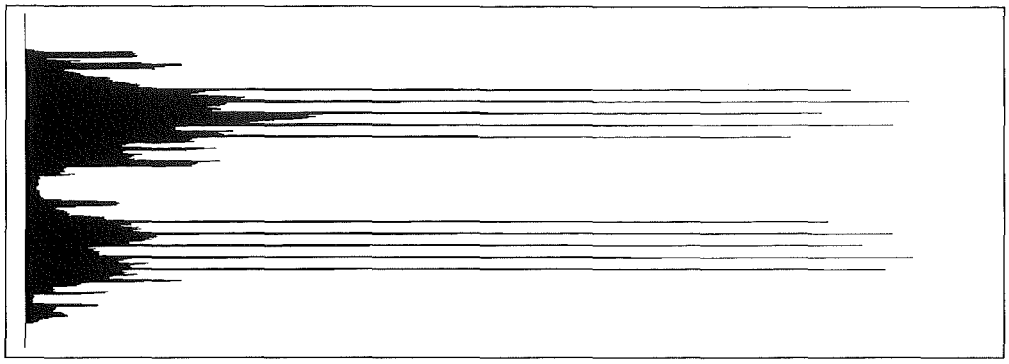


Figure 2.3: The horizontal projection of the example excerpt when scanned carefully.

2.2 Staff line identification

The definition of the problem is succinct. The OMR system is presented a bitmap, with no additional information, and must find the thickness and location of the staff lines.

The most widely used method for detecting staff lines is based on *horizontal projections* [BB92]. A horizontal projection maps a two-dimensional bitmap into a one-dimensional histogram by recording the number of black pixels in each row of the bitmap in the corresponding entry in the histogram. Under such a projection, staff lines appear as distinct peaks in the histogram that can be detected easily. Numerous variations exist, but the basic technique remains the same. The horizontal projection of the example excerpt of music is shown in Figure 2.3.

The music in Figure 2.2 was scanned carefully so the staff lines were as level as possible. It would be naïve to assume all music is scanned so carefully. The horizontal projection of a more typical scan of the example excerpt is shown in Figure 2.4. Notice how the distinct peaks in a projection deteriorate.

Instead of calculating the projection for the whole page, existing systems usually apply the basic algorithm to smaller regions, selected on the left-hand side and right-hand side of the image. This will locate short sections of staff line that are less affected by the angle at which the page was scanned. The segments can then be logically connected one-for-one, where the average gradient of these lines determines the angle of skew. Staff identification is achieved by grouping the individually detected staff lines into sets of five.

An alternative strategy for identifying staff lines is to use vertical scan lines. There is more scope for variation here. In broad terms, these algorithms use the vertical scan

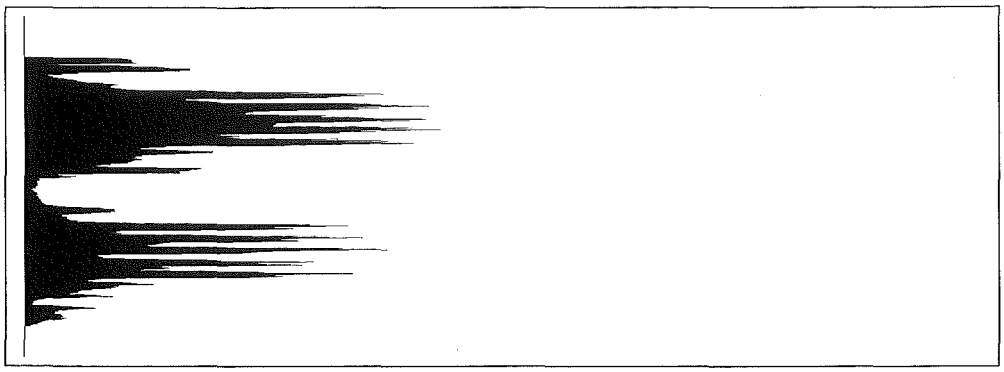


Figure 2.4: The horizontal projection of the example excerpt from a more typical scan.

lines to cut through the image, gathering data on black (or white) cross-sections. The data is then searched for a dominant feature, which corresponds to the staff line thickness (or the staff gap thickness). In addition, consistency checks, such as the identified staff lines occur at regular gaps, are often included to confirm the selected staff lines.

2.3 Object location

Once the staff lines have been identified, the individual musical objects must be located. This can be accomplished by either *ignoring* or *removing* the staff lines.

Because it is known where the staff lines are, the computer can search between the staff lines for musical objects, effectively ignoring the staff lines. When using this approach, the key task is deciding when one object stops, and the next one starts. Ignoring staff lines in conjunction with musical feature classification simplifies this decision. A search for objects between the staff lines is started, and when an object is "bumped" into, the musical feature classification stage is invoked. The result of the match directs the search for further patterns. A drawback to the approach is that as the music notation becomes more complex, so too does the control of the matching algorithm.

A more popular method is to remove the staff lines, since this is not dependent on the complexity of the music notation. The *de facto* algorithm for this task was first described by Clarke *et al.* [CBT88a]. In this algorithm, a staff line is removed piecemeal. Considering one vertical slither of a staff line at a time (working say from left to right) evidence of musical objects existing either above or below the staff line is checked for. If

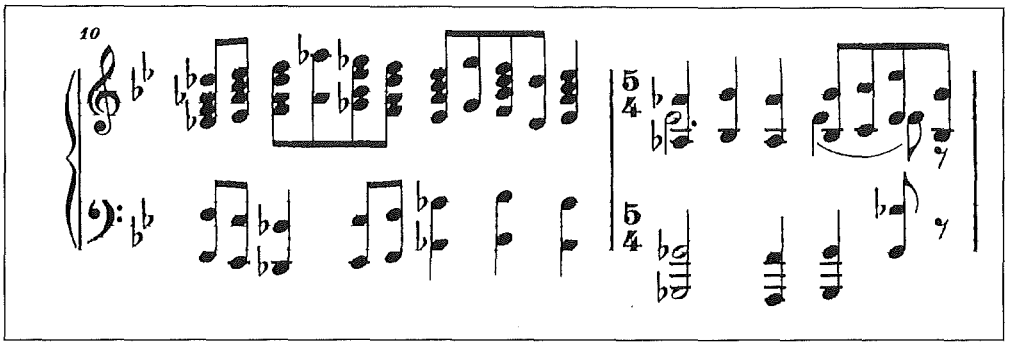
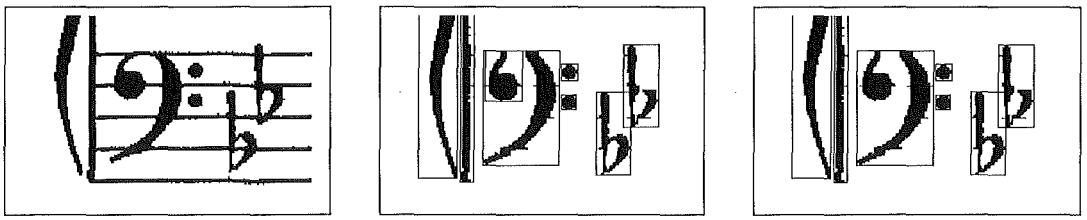


Figure 2.5: The example excerpt with staff lines removed.



(a) (b) (c)

Figure 2.6: Locating musical objects (a) the original image (b) isolated shapes broken up by staff removal (c) broken shapes reconstructed.

no evidence is found, then the slither of staff line is removed. The check for the existence of a musical object is local to a slither, searching a region no more than two pixels away from the top or bottom of that part of the staff line. Figure 2.5 shows a typical result of applying the algorithm.

Despite care being taken to keep objects whole, the algorithm does cause some fragmentation, in particular objects that blend tangentially with staff lines, such as a bass clef or a minim note head. Examples of this can be seen in Figure 2.6b, where the bass clef is fragmented in two and the second flat is broken at the top.

A simplistic solution to this problem is to join two shapes if they are situated close together, and both have an edge of their bounding box that is flush with the same staff line. The algorithm is not ideal because in certain situations it will incorrectly join objects that should remain separate, and at other times, objects that should be joined are left separate. The result of applying this imperfect algorithm to Figure 2.6b is shown in

Figure 2.6c. In this small example the algorithm correctly reforms the bass clef curl without introducing any erroneous conglomerate shapes.

By assuming that fragmentation only occurs when staff lines are removed, this simple algorithm makes no allowance for objects that become broken due to other causes. Both scanning a work that includes faint pen-work and blemishes in the original, can lead to fragmentation in any part of a shape. Thus the algorithm described is not only imperfect, but insufficient. In Chapter 6, alternative methods for handling fragmentation are investigated. Dealing with fragmented objects is an important problem that a realistic OMR system must solve.

In contrast to fragmented objects, a scanned piece may also include two or more musical objects that are joined. A distinction is made here between two classes of joined objects: objects that legitimately become joined, such as slurs and hairpin crescendos crossing bar lines; and objects that touch incorrectly, such as an accidental or the end of a slur being drawn too close to a note head. To aid clarity, the first class of joined objects will be called *superimposed*, whilst the term *touching* will refer to the latter scenario.

Strictly speaking, touching objects should never arise. Texts on the topic of music notation [Ros70, Rea74, Heu87], stress the importance of a clear, distinct layout when positioning objects, and having objects that touch reflects poor craft-work. Work in the field has indicated that reality is less than ideal, and touching objects frequently occur in printed music. Dealing with touching objects, therefore, is an important problem that a realistic OMR system must solve.

Once the musical features have been isolated (and some attempt has been made to separate touching objects and to correct any fragmentation) it is a simple step to locate individual objects. The image is processed left to right, top to bottom, systematically considering every pixel. If a pixel is found to be black, then it is used as a seed for a flood fill algorithm [FvDFH90] that also removes the shape being filled, as the algorithm proceeds. By keeping a list of all the flood-filled shapes, individual musical shapes are correctly located.

2.4 Musical feature classification

With the musical objects located, one might believe the problem of OMR has been reduced to that of OCR. Unfortunately, as Figure 1.2 on page 5 illustrated, this is not the

case. An overall observation of the differences in properties between music and text is that the shapes in music are inherently more complex. Rather than attempting to classify such intricate shapes as a whole, a trend in successful OMR systems is to work at a sub-symbol level, detecting the simpler geometric *primitive* shapes that make up symbols (such as note heads, stems and beams), and assembling them into musical features, guided by musical knowledge. Such a decomposition is often found at the heart of Document Image Analysis (DIA) systems [BBY92].

Notice how this strategy naturally deals with many of the "complications" that music has compared with text. For example, a musical feature with multiple, disjoint components (such as the key signature for A) now needs no special treatment, since it is merely an object made up of three primitive shapes (sharps in this case) where the primitive shapes are close but do not actually touch; and objects that are similar (such as a crotchet, and a quaver note), are readily distinguished by the number of individual primitives detected in that region (in this case both have a stem and a filled-in note head, but only the quaver has a tail attached to the stem).

Techniques from the area of DIA that have been successfully adapted and applied to OMR to solve the musical feature classification stage, are: Definite Clause Grammar [CC94, CBS95], Graph Grammar [FB91, FB92], Musical Knowledge with Constraints [KI90], and a Decision Tree Classifier [BD92]. All achieve musical feature classification by decomposing symbols into primitive shapes.

The process of musical feature classification is illustrated in Figure 2.7, where a labelled bounding box has been drawn around each classified musical feature. At this point, an OMR system has determined what all the shapes are graphically. Consequently a certain level of music recognition has been accomplished. From this point, it is possible to generate a "clean" version of the page, where the primitive shapes (for example note heads, and stems) can be replaced by perfectly formed geometrical shapes (for example ellipses, and lines), and if these primitive objects were correctly grouped and made available in a standard drawing editor, then the musical features could be manipulated to correct any imperfections. This in essence describes the editorial enhancement application cited in Chapter 1.

Alternatively, diatonic transposition is possible by merely shifting note heads. This style of transposition is available in the typographical notation system MusicTEX (used

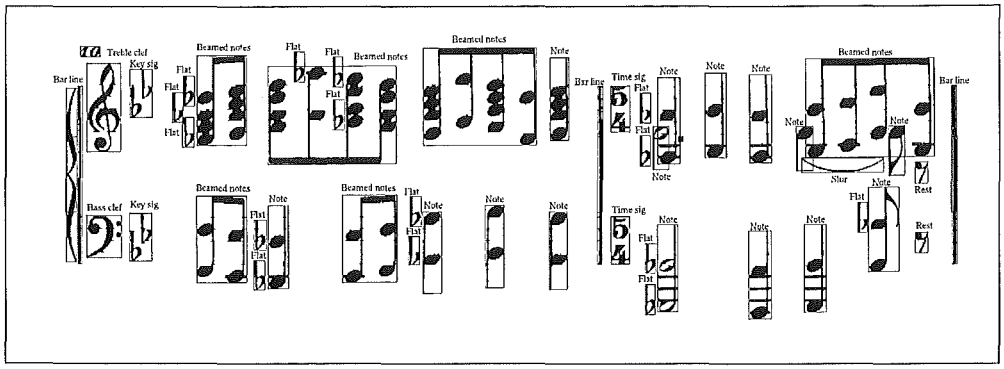
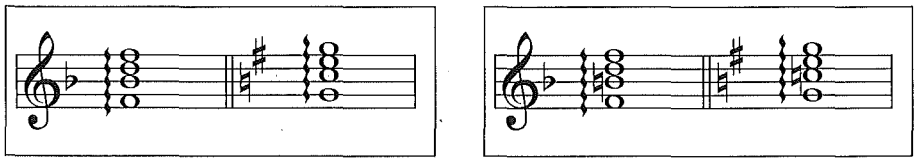


Figure 2.7: The example excerpt of music with classified musical features.



(a) (b)

Figure 2.8: Shifting note heads to transpose (a) correct transposition (b) incorrect transposition.

to typeset the music examples in this thesis), and in most situations results in the correct transformation. Figure 2.8a shows an arpeggio chord in the key of F correctly transposed into the key of G, by moving all note heads the same distance and changing the key signature. Using this method, errors in transposition occur when the tune deviates from the key indicated by the current key signature. A near identical example to Figure 2.8a is shown in Figure 2.8b, where the transposition is incorrect—the C♮ in the second chord should be written as C♯.

Musical feature classification is not, however, complete optical music recognition. Although the OMR system has graphically recognised the music, it does not yet musically understand the piece. This is why, for instance, transposition cannot be performed correctly, nor can the system play back the music as audio.

2.5 Musical semantics

The final stage in an OMR system is to extract the musical semantics from the graphically recognised shapes, and store it in a musical data structure. Essentially, this involves interpreting the spatial relationships between the detected primitives found in the image. For example, a treble clef affects the register of notes on its staff—a note head drawn on the bottom line of the staff is played as the note E, and so on. More subtle interplay occurs when the same graphical shape can mean different things in different situations. For instance, to determine if an object between two notes is a slur or a tie, the pitch of the two notes must be considered; and a dot *above* a note head changes its performance character principally by reducing its duration whereas a dot *to the right of* a note head increases the note's duration by 50%.

How this stage is accomplished is often omitted in the literature, since the operations are specific to a particular implementation. In broad terms, this phase of an OMR system is characterised by (possibly) multiple passes over a graph-like data structure, creating links, deleting links, or modifying the attributes stored at nodes due to the effect of one musical feature—say a key signature—on other musical features, such as notes.

The musical semantics of the example excerpt is shown in Figure 2.9 using an internal text format. Unfortunately there is as yet no recognised standard for representing an interpreted page of music, and hence no definitive end file format. The consequences of this in terms of the author's project are discussed in Chapter 8. For now, the best that can be achieved is a flexible musical data-structure that is easily traversed to generate specific musical file formats. Figure 2.10 shows the example excerpt reconstructed from the Tilia file format using the Lime music editor [HB93].[1]

Recent developments for a standard musical file format are promising. A consortium of interested parties (music software firms, academic researchers, and music publishers) formed a working group to address the issue, resulting in a specification for a format known as NIFF[2] (Notation Information File Format). Also, an ANSI committee has produced SMDL[3] (Standard Music Description Language) which is derived from the ISO

[1]For information about Tilia and Lime, write to Lippold Haken, CERL Sound Group, University of Illinois, Urbana, IL, USA.

[2]For information about NIFF, write to Cindy Grande, President, Grande Software Inc., 19004 37th Avenue South, Seattle, WA 98188, USA.

[3]For information about SMDL, write to Standards New Zealand, Standards House, 155 The Terrace, Wellington 6001, New Zealand, or contact the member body of ISO in your own country.

```
Staff System 1:
Staff 1:
{4Db 4F 4Ab 5Db}(0.5),{4Eb 4Ab 5C 5Eb}(0.5),{4F 4Ab 5Db 5F}(0.5),{4Ab 5Ab}(0.5),
{4Gb 4Bb 5Eb 5Gb}(0.5),{4F 4Ab 5Db 5F}(0.5),{4Eb 4Ab 5C 5Eb}(0.5),
{4Gb 5Gb}(0.5),{4F 4Bb 5Db 5F}(0.5),{4Db 5Db}(0.5),{5Eb 4Eb 4Ab 5C}(1.0) |
{4Ab 3Ab}(1.0)&4Eb(3.0),{4Bb 3Bb}(1.0),{4Ab 3Ab}(1.0),{4Bb 3Bb}(0.5)&4Eb(1.0)\_,
{4C 5C}(0.5),{5Eb 4Eb}(0.5)&_/4Eb(0.5),{3Bb 4Bb}(0.5) | .....

Staff 2:
{2F 3F}(0.5),{2Eb 3Eb}(0.5),{3Db 2Db}(1.0),{2Eb 3Eb}(0.5),{2F 3F}(0.5),
{2Ab 3Ab}(1.0),{2Bb 3Bb}(1.0),{2Ab 3Ab}(1.0) | {1Gb 2Gb}(2.0),{2F 1F}(1.0),
{1Gb 2Gb}(1.0),{2Gb 3Gb}(0.5),rest(0.5) | .....
```

Figure 2.9: Musical semantics of the example excerpt.

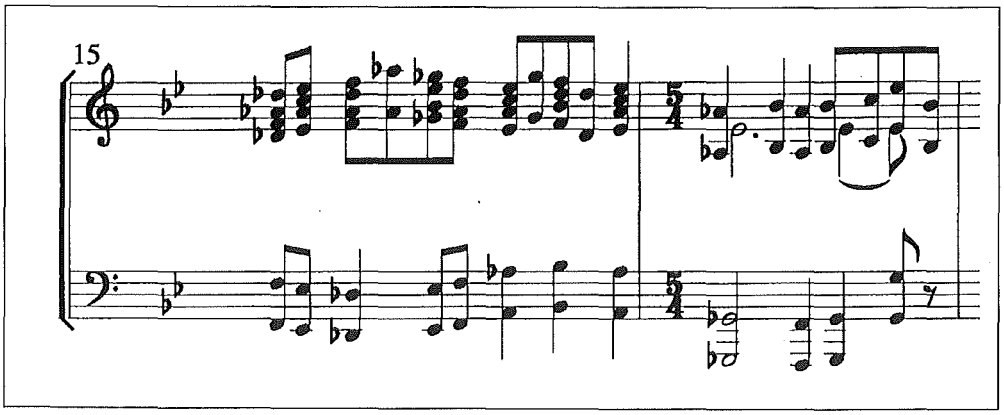


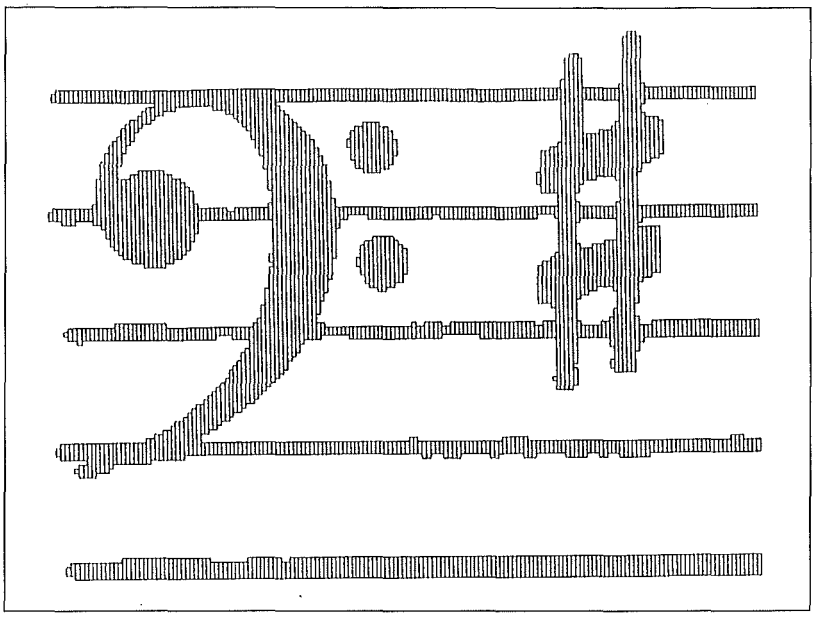
Figure 2.10: The example excerpt reconstructed in a musical editor.

standard HyTime (Hypermedia/Time-based Structuring Language). Only time will tell if either of these standards will be adopted by the computer music community.

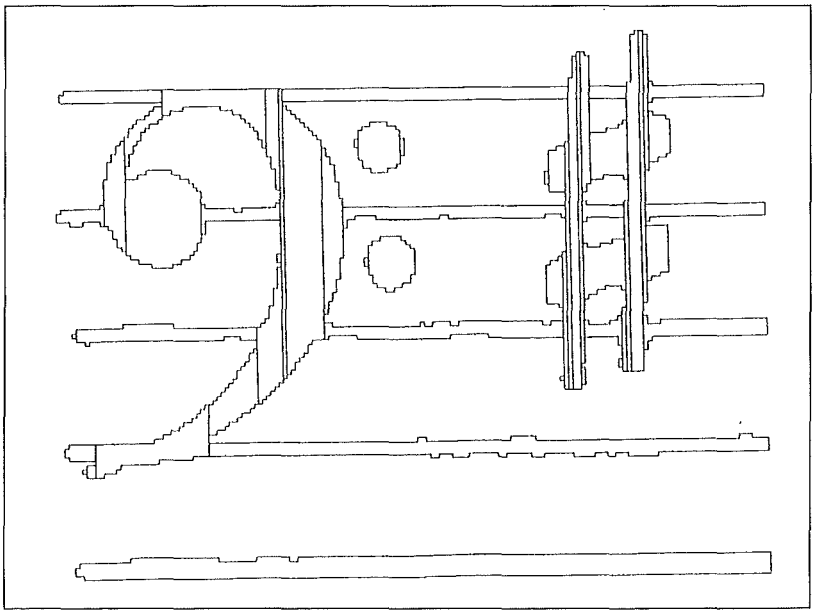
2.6 Recent trends

Early work has shown that OMR is possible by restricting the set of music notation classified, and the above discussion of a general framework to OMR has outlined general strategies to the key stages. There are, however, a range of solutions that meet the criteria.

Work by Carter [Car89, CB90, CB92, Car92a, Car93, Car94b, Car94a] is based upon a transformed Line Adjacency Graph (LAG) [Pav82]. A transformed LAG of a binary image is formed by first generating a vertical run-length encoding of the image (Figure 2.11a), then comparing the run-length encodings of adjacent columns. Runs of black that are of similar length and position in adjacent columns are continually amalgamated



(a)



(b)

Figure 2.11: Generating a transformed line adjacency graph (a) the segments formed by vertical run-length encoding (b) the segments formed by amalgamating similar vertical run-length encodings that are adjacent. (Reproduced from Carter's Ph.D. thesis [Car89]).

to form a single node in the graph, until no suitable segment exists (Figure 2.11b). To detect primitive shapes, the graph is traversed, searching for particular configurations of nodes. The system is tolerant of skew up to 10 degrees.

An interactive OMR system developed at the University of McGill by Fujinaga, has evolved steadily since 1988 [APFB88, Fuj88, FAP89, FAPB89, FPA89, FPA91, FAPH91, FAPD92, Fuj92]. The system integrates a recogniser, editor, and learner. The recogniser utilises projection methods [Fuj88] and a k-nearest neighbour classification algorithm [CH67] to detect musical features. The result is viewed using a music notation editor, where any corrections made by the user are passed on to the learning module. Here, the weights of the feature vectors used for classification are recomputed, aided by a genetic algorithm, to give an improved match next time. Even with the genetic algorithm, this is an expensive operation, and consequently the system is designed to perform the learning operation during processor idle-time.

With Fujinaga's design, the graphical classification of shapes is naturally extensible. When a musical feature is encountered for the first time it will be mis-classified. This is corrected by the operator during the interactive phase. As more examples of the new shape are encountered and corrected, the system will gradually learn how to distinguish the shape automatically.

Using a computational model from Artificial Intelligence, the system developed by Kato and Inokuchi, is a top-down approach that uses musical knowledge with constraints to recognise piano music [KI90]. The system comprises five ordered layers: image, primitive, symbol, meaning, and goal; where each layer can generate hypotheses that subsequent layers must either verify or refute. A working memory model is employed to handle communication between the layers. Based on this approach, Baumann and Dengel devised a top-down strategy that applies a decision tree classifier, and encoded musical rules to recognise piano music [BD92].

At the heart of the OMR system developed by Coüasnon *et al.* is a grammar. The primary role of the grammar is to specify the taxonomy of musical features, but it also controls the joining and segmentation of shapes as well as defining a sequential order to process objects on the staves—effectively specifying how to "read" music. The grammar forms a flexible component, making it easy to alter what constitutes valid notation.

Concentrating on the musical semantics stage of an OMR system, an attributed

programmed graph grammar [Bun82] forms the basis for the work by Fahmy and Blostein [FB91, FB92]. Traditionally a graph grammar translates an input graph into an output graph that reflects a higher level of understanding of the document. The work by Fahmy and Blostein differs slightly in that the starting point is a collection of isolated nodes that represent the primitive shapes detected in the image. The graph grammar processes these nodes, forming links that represent interactions between primitives, consequently generating a connected graph as output.

Given this variety of methods, which is best? Of course the question is too simplistic—what is best depends on the intended use of the system. An application where speed is paramount would be advised to adopt the techniques developed by the Wabot-2 project; if memory efficiency is of concern, then the methodology devised by Carter would be appropriate.

2.7 Aims of the work

Certainly no one can claim to have developed an OMR system that completely recognises any piece of music, even if the class of music was restricted to CMN. Such an aim is essentially impossible, given the variety and evolving nature of music notation. There are strict rules that must be adhered to when laying out music, however a typesetter can also include any graphical shape anywhere on the page, as long as its intention is self-evident or is explained on its first occurrence. A common situation is one where the unusual notation that indicates a particular action is repeated throughout a work. Once the musician has seen and understood the first use of the notation, other occurrences are naturally accepted when reading the remainder of the piece.

For this reason, the primary aim of this work was to develop an *extensible* OMR system. This is consistent with other OMR researchers, where extensible components to the design of OMR systems are beginning to appear. The first reported forms of extensibility came about naturally, by writing well structured code, where the set of music notation recognised could be increased by slotting additional code into place. The main drawback to this however, is that it requires expert knowledge of the source code that typically only the implementor has. Such projects in the past have petered out, once the main developer has left the research centre.

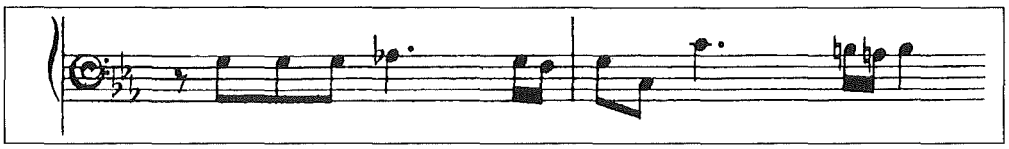
More recently OMR systems have included components that are extendible *dynamically.* By this we mean systems that can alter the set of music notation they recognise, without having to recompile the system. The systems by Fujinaga, and by Coüasnon *et al.* both have this property, though even they have their limits. In these two projects only part of the OMR system is extensible, namely musical feature classification. Since the musical semantics stage of an OMR system is dependent on the permitted graphical shapes, it is equally important that this stage is also extensible. However, neither project reports work in this area. In the former project, it is assumed that the musical meaning is dealt with in the musical editor, hence the OMR system only goes as far as graphical recognition. In the latter project, a syntactic level of recognition is achieved, though this is only part way to musically understanding the piece; by analogy, the equivalent situation in English would be knowing the type of all the words in a sentence (this word is a verb, that word is an adjective, and so on), but not knowing how they relate to one another.

To clarify the emphasis of the author's work, the primary aim was to develop an OMR system that is *totally* extensible. That is to say:

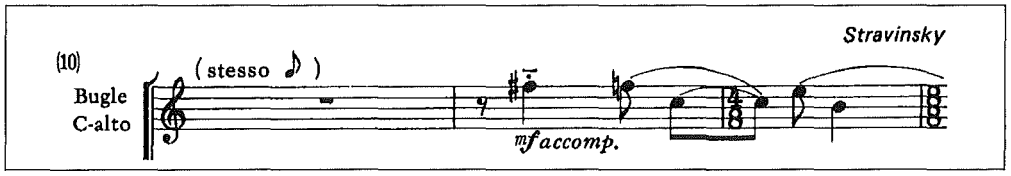
- Not restricted in the number of staff lines a staff must have.
- Not restricted to particular primitive shapes that can be detected.
- Not restricted in the valid configuration of primitives that form musical features.
- Not restricted in the musical semantics of the classified musical features.

A system that complies with these points would have a vast range of flexibility. For instance, the system would be easy to adapt to process CMN from earlier centuries, where engravers used notational practices that are no longer fashionable, such as the way the bass clef is drawn in Figure 2.12a; equally, the system would be easy to up-date to take account of newer trends such as the use by Stravinsky of a staccato dot and a stress mark to mean a sharp attack without accent, an example of which appears in Figure 2.12b. Dealing with music that includes tablature, "sol-fa" notation, and percussion—examples of which are shown in Figure 2.13—could also be accommodated. The system could even be used to recognise plainsong notation, such as the example shown in Figure 2.14.

In such an extensible system, three different types of people are involved: *the implementor*, *the configurer*, and *the end-user.* The implementor creates an extensible sys-

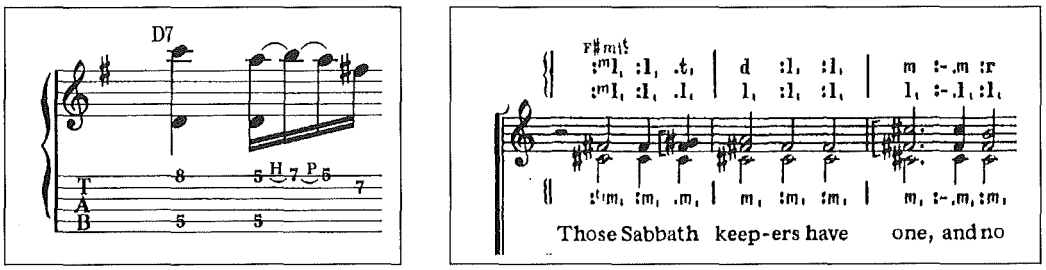


(a)

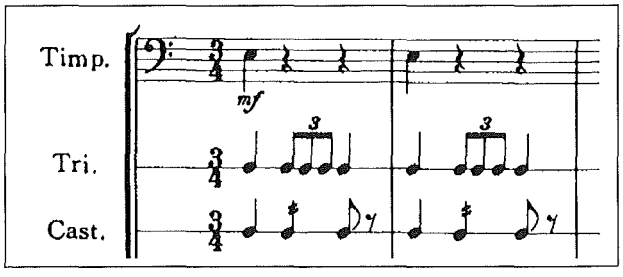


(b)

Figure 2.12: Trends in CMN shift (a) a traditional 16th Century bass clef (b) a staccato dot and stress mark are combined to form a customised articulation.



(a) (b)



(c)

Figure 2.13: Regular CMN with five staff lines often appears with other forms of music notation (a) tablature (six staff lines) (b) "sol-fa" notation for singers (zero staff lines) (c) percussion (one staff line).

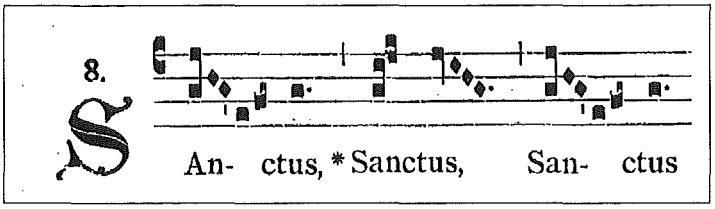


Figure 2.14: An example of plainsong notation.

tem and provides a basic set of symbols that are processed. Since it is impossible for the implementor to anticipate every need the end-user will have, the configurer is responsible for adapting the system to meet the end-user's needs.

In this thesis we will switch between these three categories where appropriate. Mostly we are concerned with the implementor's view of the problem, but in demonstrating the success of devised algorithms, we will study examples that are typical of an end-user, demonstrating the necessary configurations.

2.8 Communication between stages

At the start of this chapter a general framework for OMR was proposed (Figure 2.1). In this model, feed-back loops exist between stages, indicating that information learnt by one stage can be passed back to aid a previous stage. There is no limit on the number of iterations of this communication cycle.

This paradigm follows human behaviour. When reading music, we do not classify all the musical features first, and then deduce the musical meaning. We jump between these tasks as required. When confronted with a complex bar with split voices, we interpret the musical meaning by first classifying the musical features with note heads, and then resolving any ambiguity in the timing of the notes, because we already know the duration of the bar. Next, we might switch back to classifying musical features, and search for any accidentals or ornamentation. Of course this all happens in a blink of an eye.

Obviously, computers are not humans. And although modelling human behaviour is a worthy pursuit, to control the development of this project, the decision was made to restrict the OMR process to one-way communication links between stages, since there was sufficient work to be investigated within each stage. Such a restriction also forces inter-

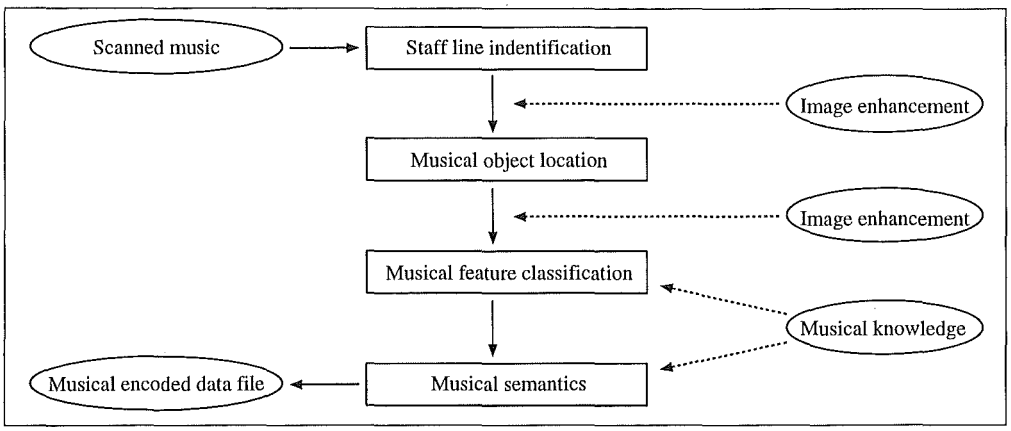


Figure 2.15: The OMR process used by this project.

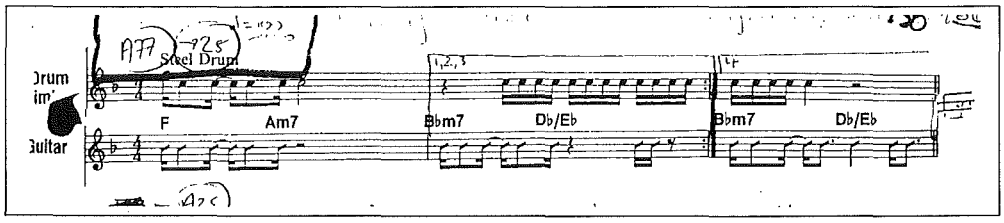
nal details in the OMR process to be exposed. This is beneficial given the nature of research work. The restricted model used is shown in Figure 2.15.

Although there is no explicit cyclic communication in the restricted model, that does not mean information learnt cannot be fed back into the system. The performance and behaviour of a particular stage can be studied by the researcher, who is then free to revise any part of the system accordingly.

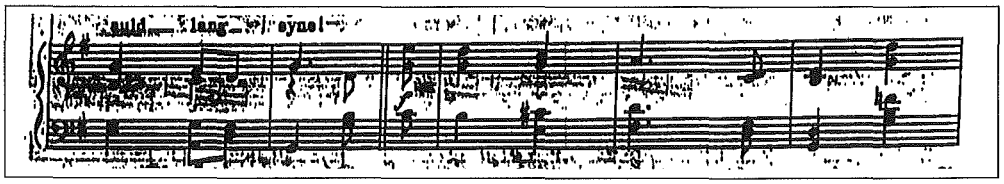
2.9 Limits on extensibility

Despite the desire to design an extensible system capable of automatically processing all forms of music notation, practical considerations must be taken into account to allow the development of an achievable system. There are some musical formats that push the requirements of the computer system beyond the scope of this project. The key attributes that vary are the *quality* of the scanned image, and the *structural rules* that the notation abides by.

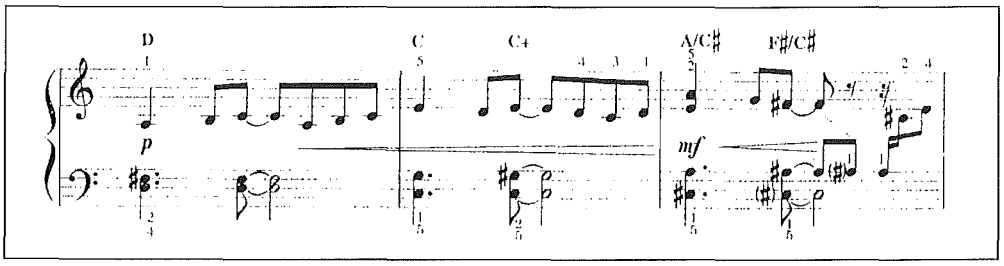
Degradation in the *quality* of the starting image affects the success of the recognition process. There are a variety of factors affecting the quality: the condition of an original manuscript, the level of generation of a photocopied work, or the thresholds used during the scanning process. Figure 2.16 shows examples of these three categories that exceed the bounds of this project. In addition to this, external circumstances control the quality of handwritten works. Figure 2.17a shows a carefully handwritten piece that appears in



(a)

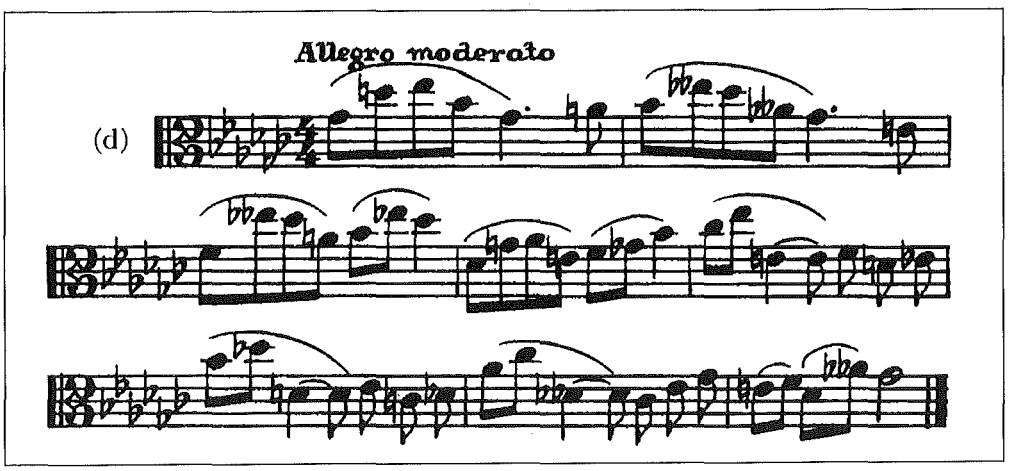


(b)

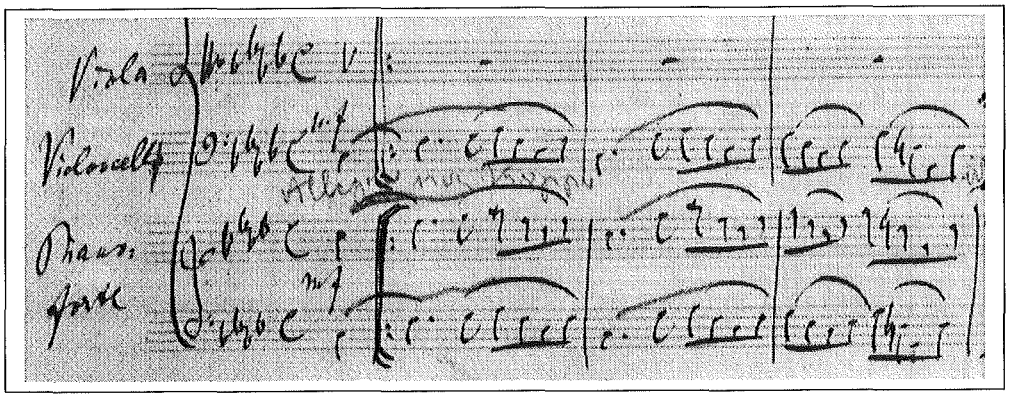


(c)

Figure 2.16: Examples of poor quality scans (a) the original work was in shoddy condition (b) the scanned page was taken from a fifth generation photocopy (c) a badly chosen threshold for intensity was used during the scan.



(a)



(b)

Figure 2.17: Extremes in handwritten music (a) an excerpt from a carefully written piece intended for publication (b) an excerpt from an original score written by Brahms.

a text book for music copyists, which our system should be able to process. In contrast, Figure 2.17b shows an original score by Brahms, which is beyond the reach of this project.

A reduction in the *structure* of the notation also affects the ability of the project to successfully process the work. Some contemporary composers find the format of CMN too constricting, and invent new ways to express their intentions. This can be done by either perturbing existing CMN or dispensing with CMN outright. Examples of this are shown in Figure 2.18. Both these examples require a level of extensibility that exceeds this project.

These limits are not severe. There is a vast body of music notation that falls within the limits of the project.

2.10 The music notation corpus

A corpus of music notation was built to test the algorithms developed. In Appendix A, a characteristic excerpt from each piece of music in the corpus is shown, along with a brief textual description.

The corpus is divided into categories using typographical criteria since this type of grouping is more useful in the analysis of OMR algorithms than a break down by historic origin, or melodic style. The categories overlap, so one piece of music may fit more than one classification. Appendix A also lists the categories each piece belongs to. Figures 2.19 to 2.22 show examples of each category.

Motivation for the categories are as follows:

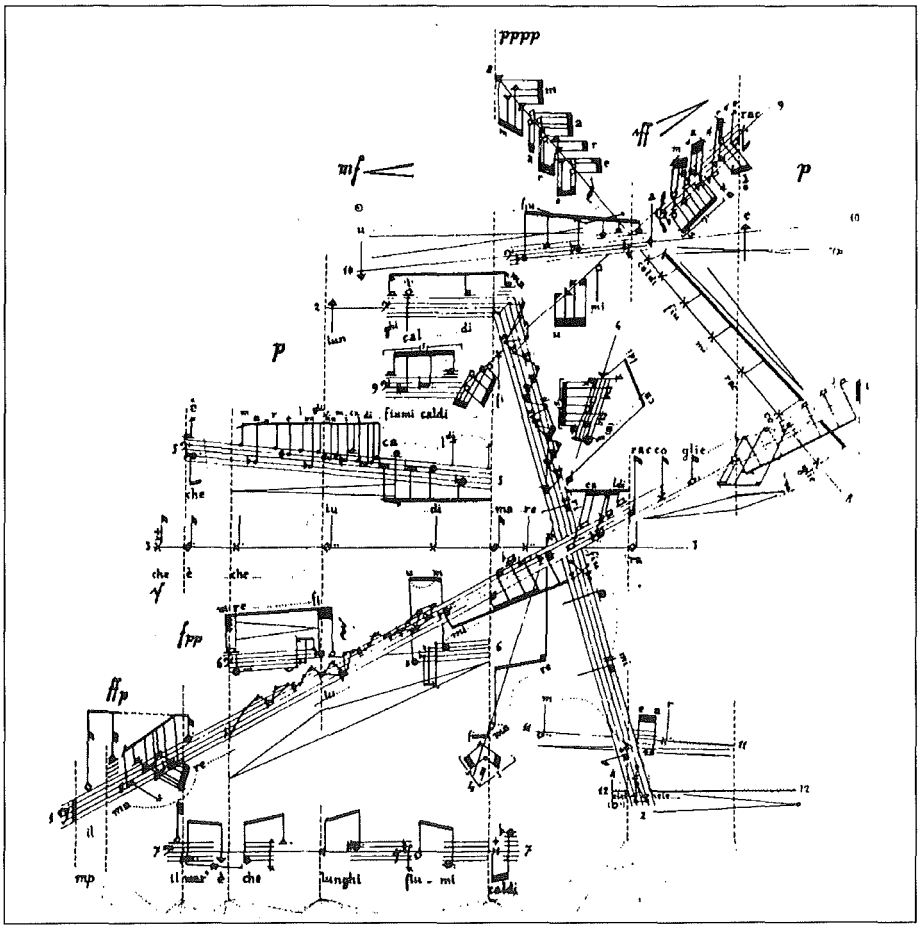
Orchestrated score. A fully orchestrated score contains large staff systems, guaranteed to test any OMR system.

Miniature score. Due to the size reduction in a miniature score, the notation is harder to follow, however it *is* still possible to understand—a good test for the lower resolution limit of an OMR system.

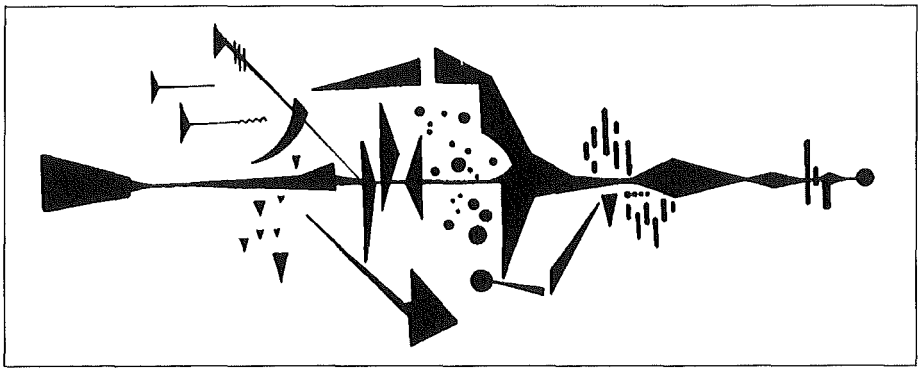
Accompaniment. This is typically a piano piece that includes the solo instrument part drawn on a smaller staff. Such music will show if an OMR system has a good decompositional design.

Monolinear music. This term defines music with no chords within a staff.[4] This cate-

[4]The term *monolinear* was first coined by Kassler [Kas72].

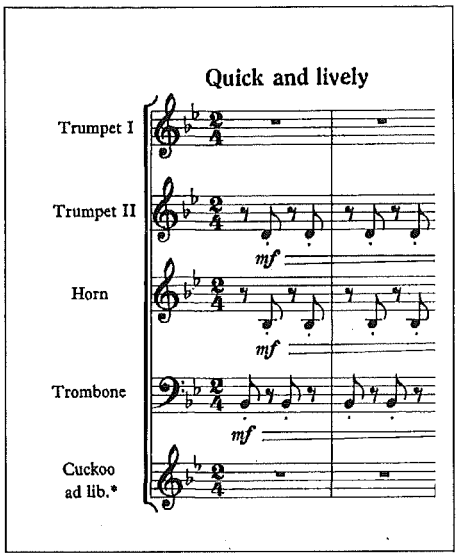


(a)

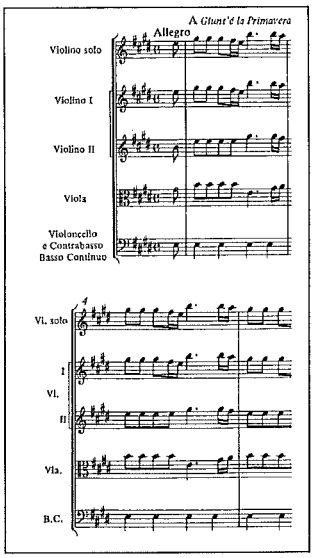


(b)

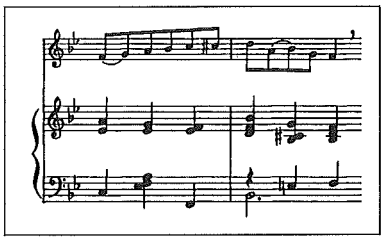
Figure 2.18: Composers can find CMN too restrictive (a) "Siciliano" by Sylvano Bussotti perturbs existing CMN (b) "Four Visions (No. 2)" by Robert Moran dispenses with CMN outright.



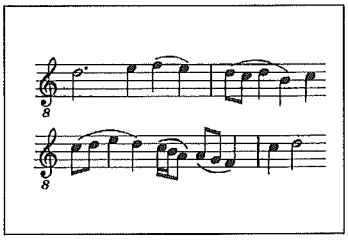
(a)



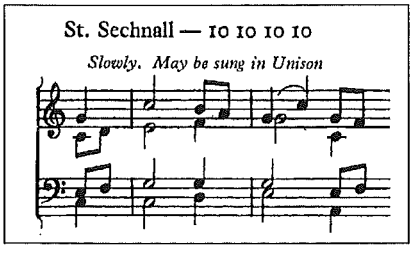
(b)



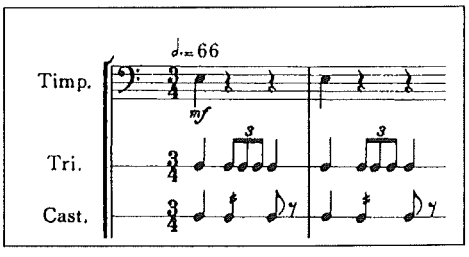
(c)



(d)

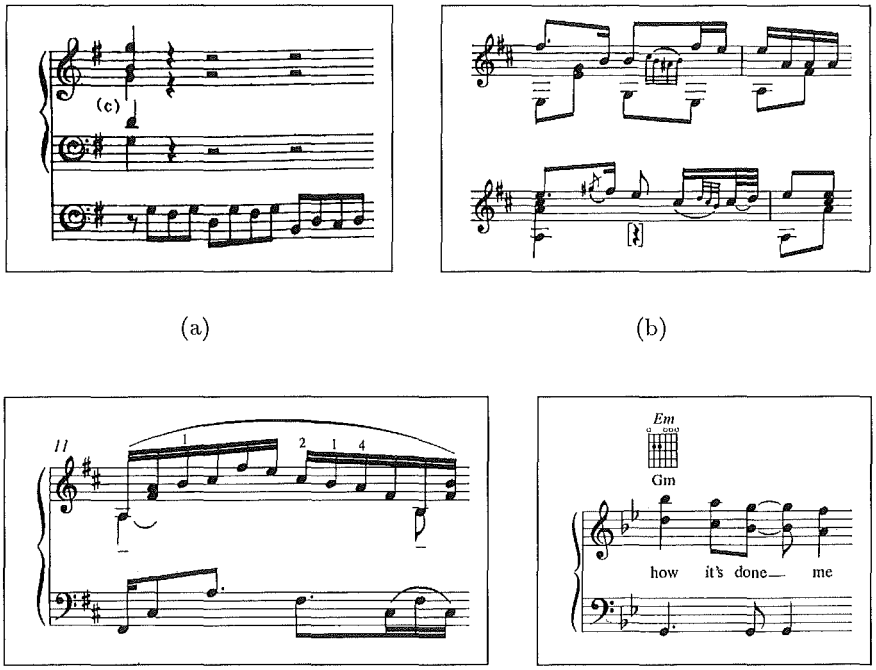


(e)



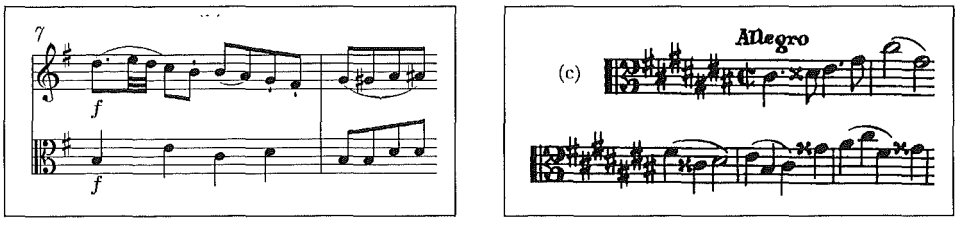
(f)

Figure 2.19: Examples of CMN from the corpus (a) score (b) miniature score (c) accompaniment (d) monolinear (e) hymn (f) percussion.



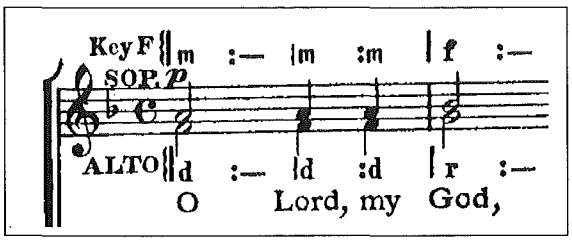
(a) (b) (c) (d)

Figure 2.20: Examples of individual instrument parts from the corpus (a) organ (b) guitar (c) piano (d) popular piano.

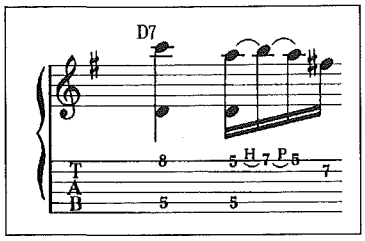


(a) (b)

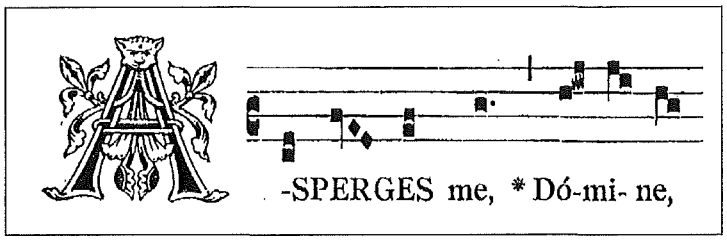
Figure 2.21: Examples of non-engraved CMN from the corpus (a) computer typeset (b) handwritten.



(a)



(b)



(c)

Figure 2.22: Examples of less common music notation from the corpus (a) sol-fa (b) guitar tablature (c) plainsong notation.

gory forms a simple collection of CMN music that a basic OMR system should be able to process.

Individual instrument. Music for specific instruments include special notation, such as bowing marks for violin music. This collective category is further divided by instrument. The corpus includes the sub-categories: **guitar**, **violin**, **organ**, **piano**, and **popular piano**, though the sub-division is by no means comprehensive. The popular piano sub-category refers to the common style of piano music that includes the lyrics, the guitar chords, and optionally the melody line, along with the piano part. An example is shown in Figure 2.20d. The individual instrument category tests the extensibility of an OMR system.

Hymn. The most common source for hymn music is from tightly-bound books that use thin paper. Often the music was typeset in the 19th Century. This all adds together to form scanned music of a poorer quality, and provides a good test for the reliability and robustness of an OMR system.

Percussion. Most percussion uses uncommon symbols. Unpitched percussion is written using a single staff line. Both test the extensibility of the OMR system.

Computer typeset. Computer music editors are increasing in sophistication and popularity, but currently their automatic rules for layout can produce less than ideal results. Ironically, therefore, it is a good test for the reliability and robustness of an OMR system. In addition to this, the scanned image can be produced without ever scanning it! This is convenient if you wish to test part of an OMR system free from the effects of skew and noise.

Handwritten. In handwritten music, the primitive shapes become less reliable. An OMR system needs to be flexible and tolerant of deviations to process this category of image.

Sol-fa. This singing notation is text based and uses no staff lines. It is another test of how far the extensible design of an OMR system goes.

Tablature. This notation is suited for fretted stringed instruments, where numbers appear on the staff lines, rather than notes. In its most common form—for guitar—a staff has six staff lines and normally appears with a CMN version that has difficulty conveying the same information. Another extensibility test for the OMR system.

Plainsong notation. This notation uses four staff lines and neumes—a grouping of notes that corresponds to one syllable of text. It is a style of notation substantially different from CMN. This tests the extensibility of an OMR system, particularly the musical semantics stage. There is also a diverse range in the quality of scores, which again tests the reliability and robustness of the OMR system.

Although the categories in the corpus support a wide range of music, representative items were restricted to ones we would expect an OMR system capable of processing. Thus the corpus is where the boundary of acceptable music notation for this project is specifically defined. For instance, the handwritten music category is not arbitrary handwriting. It is restricted to good quality handwritten notation, such as that shown in Figure 2.21b, destined for presentation in musical texts.

Category	Representative piece
Orchestrated score	"The Cuckoo"
Miniature score	"La Primavera"
Accompaniment	"Bonita"
Monolinear (no chords)	"Music Reading: Exercise 1"
Hymn	"Hymns: Ancient & Modern"
Percussion	"Music Reading: Exercise 5"
Individual instrument (organ)	"Prelude and Fugue in G Major"
Individual instrument (guitar)	"Introduction et Variations: Sur un Motif de Rossini"
Individual instrument (piano)	"Big My Secret"
Individual instrument (popular piano)	"Hazard"
Computer typeset	"Duetto uno a Violino e Viola"
Handwritten	"Elementary Training for Musicians: Exercise 1"
Sol-fa	"Parry My Soul"
Guitar tablature	"Blues Break 1"
Plainsong notation	"Extra Tempus Paschale"

Table 2.1: The title of the representative piece from each category.

2.11 A representative cross-section

The examples shown in Figures 2.19 to 2.22 form a representative cross-section of the corpus that will be used in subsequent chapters to test algorithms. The title of the representative piece from each category is listed in Table 2.1.

Chapter 3

Staff detection

Detecting where the staff is—whether it be five lines, four lines, or one line in number—is crucial to OMR. In this chapter, existing algorithms for staff detection are surveyed and discussed. From this, deficiencies are highlighted, and an improved strategy proposed.

The new strategy is decomposed into five steps, and in implementing each step, choices—such as what threshold value to use—must be made. Rather than making these decisions arbitrarily, a representative cross-section of the corpus (one image from each category) was taken, and various plausible configurations for each step were tested. At the end of the chapter the final algorithm is formed by taking the best configuration for each step, and the results from processing the opening page of every entry in the corpus with this algorithm are presented and analysed.

3.1 Existing techniques

The central task in staff detection is the identification of staff *lines*. Existing work can be divided into two categories: *vertical* methods and *horizontal* methods. These terms are perhaps misleading, because both strategies also require work in the orthogonal axis. They take their names from the initial direction in which each algorithm proceeds, since this dominates the methodology.

Vertical methods differ substantially in detail. In general though, they start by taking a series of vertical scans through the image, and for each scan, identify potential sections of staff line. Consistency checks are then made horizontally across the scan lines to precisely determine the staff lines. These steps are shown in Figure 3.1a. In contrast,

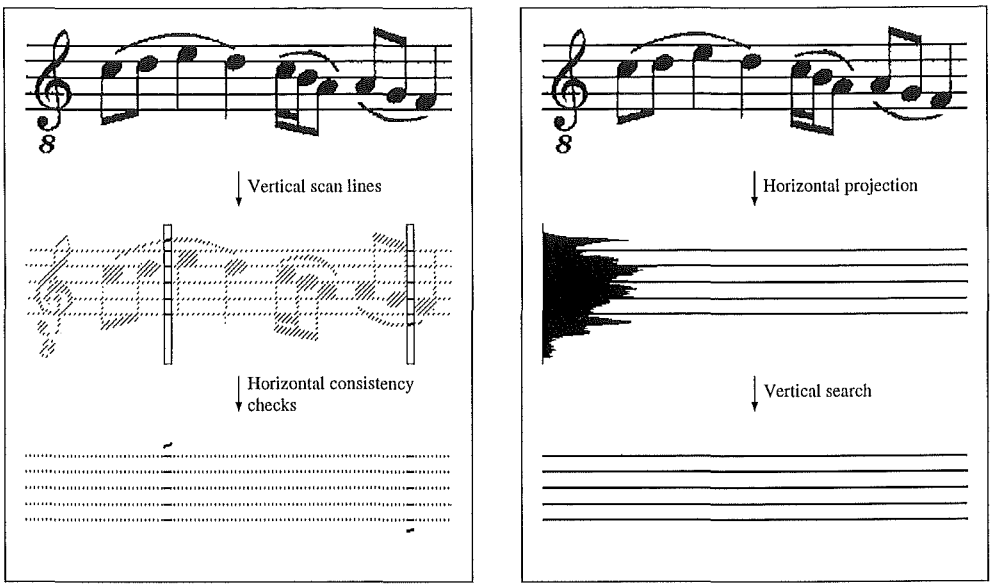


Figure 3.1: Overview to methods that detect staff lines (a) vertical based method (b) horizontal based method.

horizontal methods differ little in technique. They start by forming the horizontal projection of the image, which is then searched vertically for distinctive peaks. These steps are shown in Figure 3.1b.

3.1.1 Vertical methods

There are four reported OMR systems that detect staff lines using a vertical scan based algorithm.

The approach developed by Kato and Inokuchi [KI90] starts with 10 evenly spaced vertical scan lines to estimate the global thickness of the staff lines and the staff gaps. For each scan line, run-length encodings of both the black and white pixels are generated. The highest value in the histogram for black run-length encodings is interpreted as the average staff line thickness, and the highest value in the histogram for white run-length encodings is interpreted as the average staff gap. Using this information, the vertical scan lines are re-analysed to obtain local approximations for the location of the staves. These estimated staff areas are further targeted by applying a rectangular mask to the left-hand side and right-hand side of the area, in which short components are eliminated. A verti-

cal projection of each filtered region is used to home-in on the start (or end) of the staff, and a horizontal projection is then used to obtain accurate values for the staff line thickness, staff gap, and staff height.

In the method described by Glass [Gla89], intensive groupings of vertical scan lines are taken around the middle of the left-hand half of the page, and the middle of the right-hand half of the page. For each group of vertical lines, consistency checks locate five lines of equal thickness with four staff gaps of equal thickness between them. The issue of automatically determining the size of music notation was not resolved; instead the user is required to enter a rough estimate of the staff gap at the beginning of the process. This could easily be replaced by the highest histogram value method devised by Kato and Inokuchi.

The method devised by Reed [Ree95] starts by taking vertical scans at regular intervals.[1] Next the vertical scans are searched for *staff samples*—five equally spaced, equally sized run-lengths of black pixels—which are then collected into *staff groups* if they are of similar spacing, run-length thickness, and vertical position. Finally, the median value of adjacent staff samples in the various staff groups is calculated as an approximation for the angle of skew contained in the image. The process includes false staff samples that must be eliminated. An additional step, therefore, is to apply the Hough transform—a method for evaluating the likelihood that a specified parametric equation exists at a specific place in the image [BT88]—to the data to find the dominant lines that exist in each staff group.

The decision by Carter to use a transformed Line Adjacency Graph (LAG) (Section 2.6, page 26) naturally led to a vertical based solution for identifying staff lines [Car89, CB90]. In his solution, the transformed LAG is searched for potential sections of staff line (called filaments), determined by nodes in the graph that have a high aspect ratio, a low curvature, and are singly connected. Because, at this stage in the system, it is not known how thick a staff line should be, other shapes—such as beams, or hairpin crescendo and diminuendo markings—are likely to pass this test. To filter these out, a histogram of filament thickness is used to set a maximum thickness for a staff line, and the erroneous shapes are eliminated. Filaments are further filtered by checking for collinear sections, and successful candidates are joined together, bridging the gaps left by musical features, to form the staff lines. Finally, the staves are detected by locating groups of five

[1] Reed found a consistent spacing of approximately 0.1 inch (0.25 cm) worked well.

staff lines that overlap horizontally, and whose vertical spacing of adjacent staff lines is roughly equal to the estimated staff space height.

3.1.2 Horizontal methods

Only one basic horizontal algorithm for identifying staff lines has been described in the literature. Based on horizontal projections, the general form of the algorithm has already been presented in Section 2.2. Implementations differ slightly in minor details, such as how the region on the left-hand side and the region on the right-hand side are chosen for the restricted projection, or how the threshold is set for peak detection.

The Wabot-2 project, which utilises a low-pass filter to detect staff lines, initially appears to be using a new approach, but this is simply different terminology for a hardware implementation of the same horizontal projection technique. Where this solution *does* differ from the norm is in the number of short staff segments located. Rather than relying on two projected regions for locating staff lines, and assuming that the staff lines are straight, the Wabot-2 project calculates the projections of contiguous regions across the page. Thus, any deviation from the expected straight staff line is recorded.

3.1.3 Detecting staff systems

A staff system is a collection of staves joined at each end by bar lines, and indicates that each staff should be played concurrently. Normally staff systems are stacked vertically, but a musical score can include staff systems that are horizontally separated, as shown in Figure 3.2. There is no evidence in the published literature that this situation is processed correctly. As described, the algorithms would detect *one* staff system with three staves, rather than *two* staff systems each with three staves. In these algorithms, no account has been made for separate staff systems existing side-by-side.

A simple refinement is to first detect the rectangular regions in which staff systems exist, and then apply the basic staff line detection algorithm to each region—just as the horizontal projection method was applied to restricted regions of the image to cope with skew.

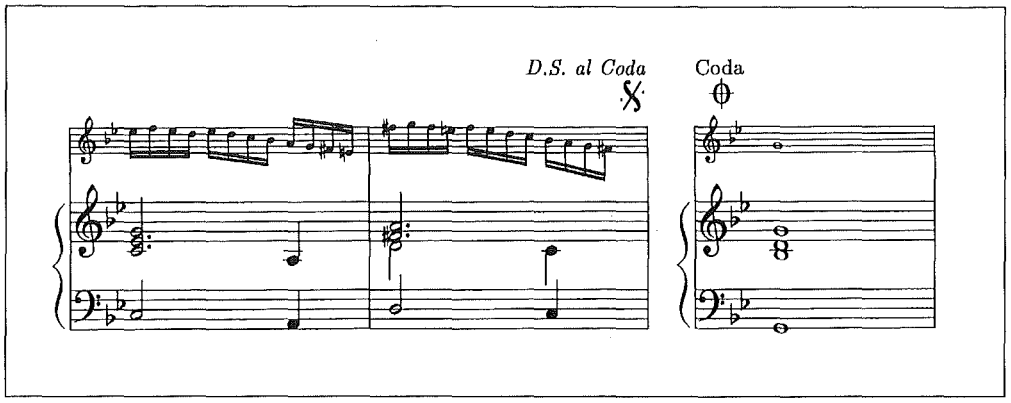


Figure 3.2: An example excerpt of music with a coda, and different sized staves within one staff system.

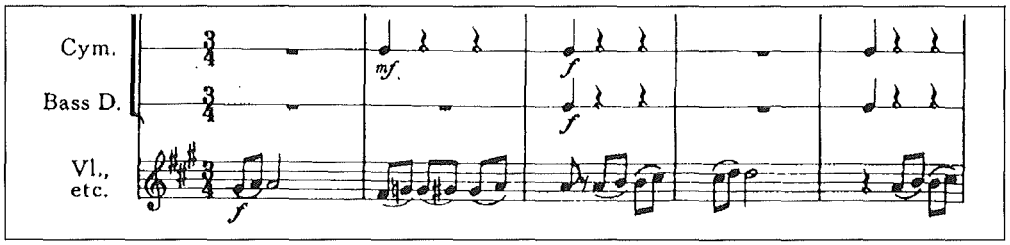


Figure 3.3: As described, existing vertical based staff detection methods fail to correctly locate staff systems consisting of one staff line.

3.1.4 Comparing vertical methods and horizontal methods

A vertical scan based approach to staff line identification is attractive because it trivially copes with the likely situation of a scanned image that is skew. A horizontal approach, on the other hand, is complicated by skew, and requires a more convoluted algorithm to address this issue. Moreover, Glass reported that his vertical based solution was, on average, three times faster than the standard horizontal approach [Gla89].

Unfortunately, there is a draw-back to the vertical approach. All the described vertical methods assume a staff consists of more than one line, relying on the existence of a staff gap to function correctly. This, however, excludes music with unpitched percussion, such as the example shown in Figure 3.3. It might prove possible to adapt such algorithms to handle this class of music, but there is no discussion of this in the literature. We therefore turn to the horizontal projection method as a basis for a new and improved staff detection algorithm, since this method will naturally detect a single horizontal line.

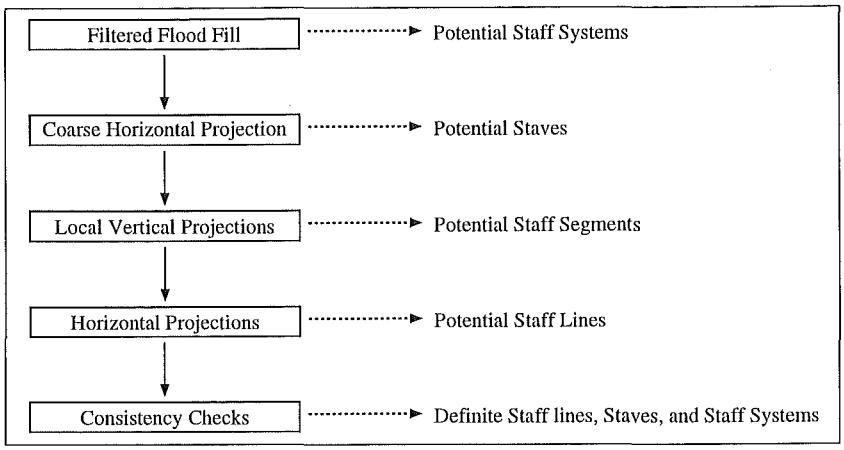


Figure 3.4: A generalised algorithm for staff detection.

3.2 A new approach

Figure 3.4 illustrates an enhanced algorithm for staff detection. The procedure includes the same steps used in existing horizontal methods, but it also includes some new steps. The main alteration is to remove the restriction that staves consist of five staff lines. The strategy also processes staff systems of variable size, which many existing algorithms have overlooked.

First, all the connected objects on a page are located using a flood-fill algorithm that filters the objects so that only likely staff system candidates remain. Next, a horizontal projection for each object is taken, and sections of intense peak activity in the histogram are marked as potential staff areas. A vertical projection over each potential staff area is then used to locate segments where one would expect to find only staff lines, based on low activity in the projection. Choosing two suitable segments—one on the left-hand side, and one on the right-hand side—a horizontal projection is taken for each segment, and the detected peaks are recorded as potential staff lines. Finally, consistency checks on the possible staff line segments are performed to make the decision definite. From this, the actual staves and staff systems are deduced.

A complication that arises due to the relaxation in the number of staff lines a staff may have, is that it is possible for a line of text to be mis-classified as a staff. Western text uses three prominent construction lines: the top of capital letters, the top of lower case letters, and the base-line (Figure 3.5a). In a horizontal projection, these areas build

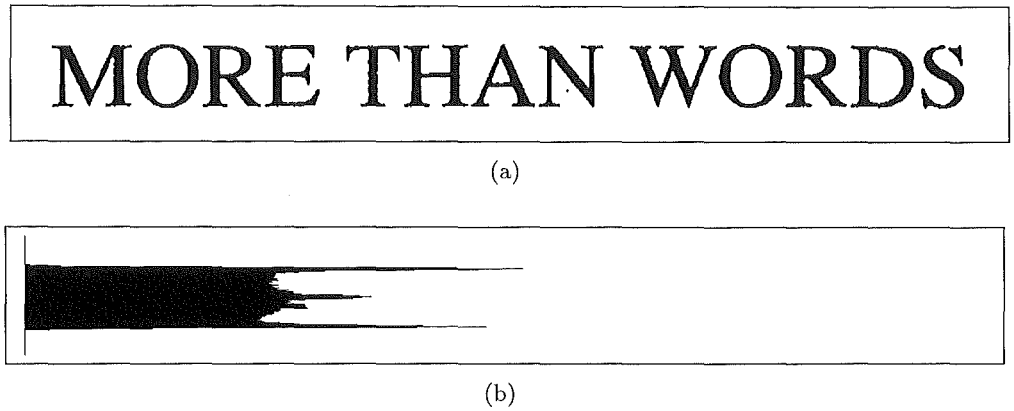


Figure 3.5: Western characters use three distinct heights within a line of text (a) the title to a piece of music (b) its horizontal projection.

up to form peaks—a situation that is exacerbated by fonts with serifs, which are common in titles for music (Figure 3.5b). Small, false staff segments can be found that cut through letters, resulting in staff lines being detected. The consistency check for these false staff segments finds two or three thin lines of equal spacing, and consequently the region is classified as a staff.

One solution to this problem is to insist that a staff contains four or more staff lines. This would correctly process a large body of music, including plainsong notation, pitched CMN, and tablature. Unfortunately, it rules out CMN that includes unpitched percussion, since this uses a single line per staff. A better solution is to introduce an *OCR preprocessing step* to the system, that detects and removes all text before staff detection is performed.

This means that we are now applying OCR to an image before any attempt has been made to correct the image for skew. Fortunately, it is not crucial that the OCR package can cope with this situation. A preliminary pass of the staff line detection algorithm can be used to detect the angle of skew, since the false staff lines caused by text will not affect this calculation. Using this information the image can be safely rotated to eliminate the skew, allowing the OCR package to be applied to a corrected image, removing the text before the staff detection algorithm is re-applied to find the true staves.

The five stages of the algorithm shown in Figure 3.4 are now explained in detail, using the excerpt of music shown in Figure 3.2 as an example.

3.2.1 Potential staff systems

Potential staff systems are located using the same methodology that was previously described to isolate musical objects once the staff lines have been removed (Section 2.3, page 22). Each pixel in the image is systematically considered, and used as a seed to a flood-fill algorithm if it is coloured black. The located objects are then divided into two lists: potential staff systems, and non staff systems, based on the object's width and height. At this stage in the algorithm, the decision about whether or not an object is a staff system need not be precise, but the objects that are chosen as potential staff systems must form a superset of the real staff systems.

Flood-fill variations

The basic flood-fill routine recursively visits all the black neighbouring pixels of the pixel currently under consideration that have not previously been visited, until there are no pixels left to visit [FvDFH90]. Two variants were developed: 2-pixel horizontal connectivity and 8-pixel horizontal connectivity, to compensate for potentially broken staff lines. The 2-pixel adaptation considers neighbouring pixels up to two pixels away in the horizontal direction; similarly the 8-pixel version considers eight pixels either side of the current pixel. With the latter variant, a convenient approximation is to visit a neighbouring *byte* if its value is non-zero. Such an adaptation avoids repetitive bit extractions.

Using these variants, objects that are not physically joined by black can be located as a single object. This is a desirable property, since staff lines are thin and prone to breaking up. However, it is important that independent staff systems stay separate. In the worst case scenario of using the byte-based 8-pixel horizontal connectivity algorithm on a piece of music scanned at 150 dpi—generally agreed to be the lowest resolution that OMR can be expected to go—the largest jump the algorithm could make is 14 white pixels. This is equivalent to 0.09 inch (0.24 cm). It is hard to imagine a piece of music where staff systems are placed this close together. In the corpus, the smallest horizontal staff system separation is 0.30 inches (0.77 cm).

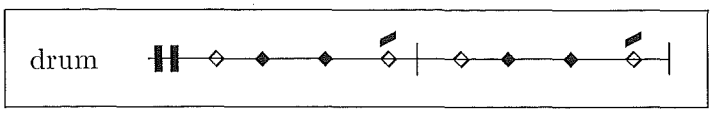


Figure 3.6: Minimum conceivable height for a staff system.

Measurement	Threshold
Minimum width	$0.75 \times dpi$ pixels
Minimum height	13 pixels

Table 3.1: Minimum width and height for potential staff systems.

Thresholds

The filter for potential staff systems was based on a minimum width and height for a staff system that an object must pass to be considered. How should these limits be set?

When considering a piece of music, the shortest conceivable staff system *width* would be due to a coda. This must include a clef, and at least one bar of notes. From a survey of works in the corpus, the minimum width for a potential staff system was set at 0.75 inch (1.90 cm). This threshold easily includes all staff systems in the corpus,[2] and therefore it is conjectured that this value is suitable for other scores.

The minimum *height* for a potential staff system can be based on the staff line thickness. An extreme situation for an OMR system is one where the scanned image contains staff lines that are one pixel thick—beyond this it would be unreasonable to expect an OMR system to perform reliably. The smallest conceivable height for a staff system would be one consisting of a single staff of unpitched percussion, where the bar line is the tallest object (Figure 3.6). Combining these two facts, the minimum height for a potential staff system was set at 13 pixels. A survey revealed that all staff systems in the corpus were higher than this value. Likewise to the minimum width threshold, this value is liberal in what may pass as a staff system, and consequently it is anticipated that future scanned images will safely pass this test.

The thresholds used for minimum width and height are summarised in Table 3.1. The choice of these thresholds is not critical to the overall success of the algorithm. In fact, a minimum threshold of zero for both width and height would meet the requirement

[2]The smallest staff system width in the corpus is 2.70 inch (6.85 cm).

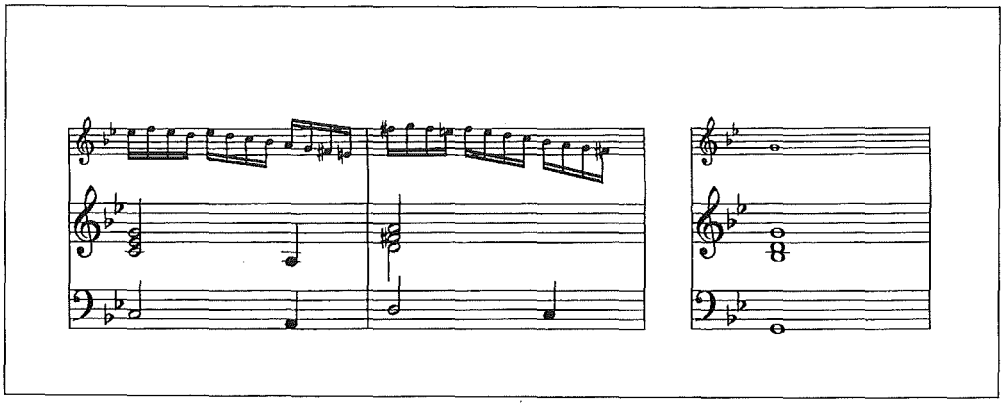


Figure 3.7: The potential staff systems found using the 1-pixel or 2-pixel flood-fill algorithm.

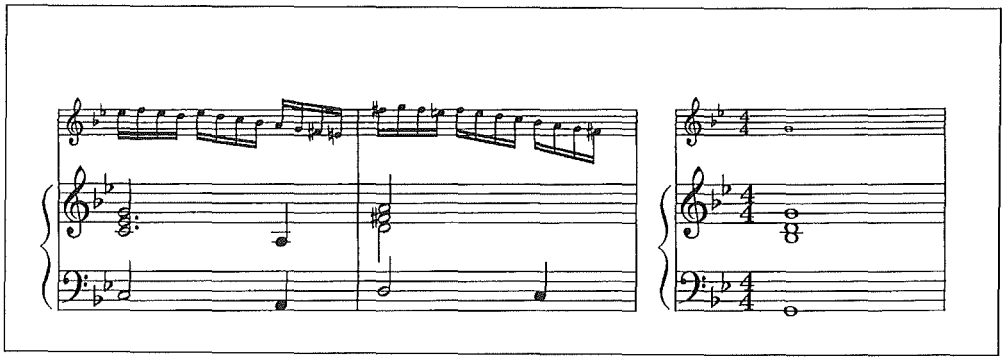


Figure 3.8: The potential staff systems found using the byte-based 8-pixel flood-fill algorithm.

that the selected objects must be a superset of the real staff systems. The chosen settings improve the efficiency of the algorithm by ruling out the numerous small markings contained in a page of music that are clearly not staff systems.

Evaluation

The three filtered flood-fill algorithms were applied to the scanned image shown in Figure 3.2 on page 47. For this image, both the 1-pixel and the 2-pixel algorithms resulted in identical potential staff systems being detected (shown in Figure 3.7). The same staff systems are detected by the byte-based 8-pixel algorithm, except that this time the two located shapes each include a brace mark (shown in Figure 3.8). The byte-based 8-pixel algorithm is prone to including such nearby shapes, but this does not pose a problem since these shapes are correctly isolated once the staff lines have been removed.

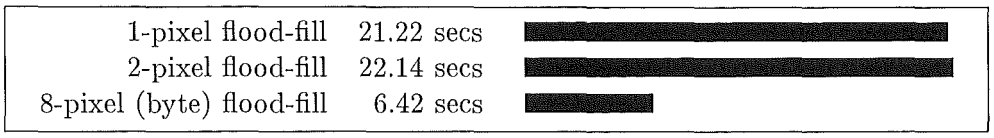


Figure 3.9: Average time taken to process the representational cross-section of the corpus for potential staff systems.

In a more extensive test, the three algorithms were used to process the representational cross-section taken from the corpus. The results are shown in Table 3.2. Both the 1-pixel and 2-pixel flood-fill algorithms fail to include four of the staff systems present in the plainsong notation example, due to small breakages in the staff lines. In contrast, the 8-pixel byte-based flood-fill algorithm correctly includes all the staff systems in all the images, but also includes a higher number of erroneous staff systems than the other two algorithms.

Using the byte-based algorithm, objects with only a small horizontal separation between them are detected as one large object. Sometimes the amalgamated object exceeds the minimum width requirement for a potential staff system, thus increasing the number of potential staff systems detected. Fortunately, these erroneous staff systems are ruled out by subsequent processing steps.

The average time taken to process these images is shown in Figure 3.9. Overall, the byte-based 8-pixel flood-fill algorithm is 3.3 times faster than its competitors, and is therefore selected as the best algorithm for this step in the staff detection algorithm.

3.2.2 Potential staves

An important concept in locating potential staves is the notion of a *staff area*—a rectangular area within a potential staff system where there appears to be a high activity of horizontal lines. Figure 3.10 details the steps involved in accomplishing this task. The essential value that needs to be computed is the *staff jump*, which is the minimum distance between one staff and the next.

For each potential staff system, its horizontal projection is generated (Figure 3.11), and all the peaks higher than half the height of the tallest peak in that staff system are gathered (Figure 3.12). Next, the vertical separation between adjacent peaks is calculated and sorted into ascending order (Figure 3.13). This list is then searched for an item that

Category	Number of staff systems	Number of staff systems missed	Number of extra staff systems
Orchestrated score	3	0	2
Miniature score	2	0	0
Accompaniment	4	0	2
Monolinear (no chords)	12	0	0
Organ	4	0	0
Guitar	8	0	15
Piano	4	0	21
Popular piano	6	0	3
Hymn	4	0	0
Percussion	1	0	0
Computer generated	6	0	9
Handwritten	10	0	4
Sol-fa	4	0	0
Tablature	6	0	0
Square head notation	5	4	17
Total	79	4	73

(a)

Category	Number of staff systems	Number of staff systems missed	Number of extra staff systems
Orchestrated score	3	0	2
Miniature score	2	0	0
Accompaniment	4	0	2
Monolinear (no chords)	12	0	0
Organ	4	0	0
Guitar	8	0	15
Piano	4	0	21
Popular piano	6	0	3
Hymn	4	0	0
Percussion	1	0	0
Computer generated	6	0	9
Handwritten	10	0	4
Sol-fa	4	0	0
Tablature	6	0	0
Square head notation	5	4	21
Total	79	4	77

(b)

Category	Number of staff systems	Number of staff systems missed	Number of extra staff systems
Orchestrated score	3	0	14
Miniature score	2	0	7
Accompaniment	4	0	13
Monolinear (no chords)	12	0	1
Organ	4	0	5
Guitar	8	0	20
Piano	4	0	21
Popular piano	6	0	2
Hymn	4	0	3
Percussion	1	0	3
Computer generated	6	0	16
Handwritten	10	0	5
Sol-fa	4	0	12
Tablature	6	0	1
Square head notation	5	0	17
Total	79	0	140

(c)

Table 3.2: Accuracy of the potential staff systems algorithm when applied to the representational cross-section of the corpus (a) 1-pixel flood-fill (b) 2-pixel flood-fill (c) 8-pixel (byte) flood-fill.

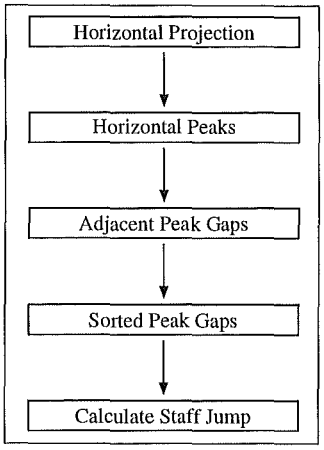


Figure 3.10: The steps used to determine the staff areas within a staff system.

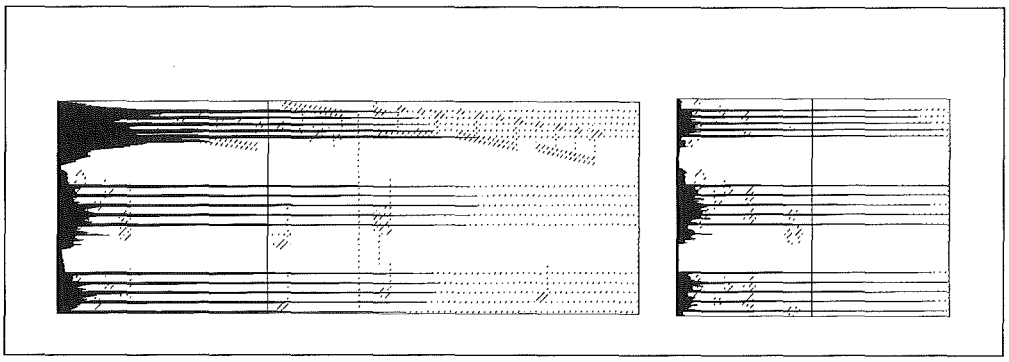


Figure 3.11: Horizontal projections superimposed on potential staff systems.

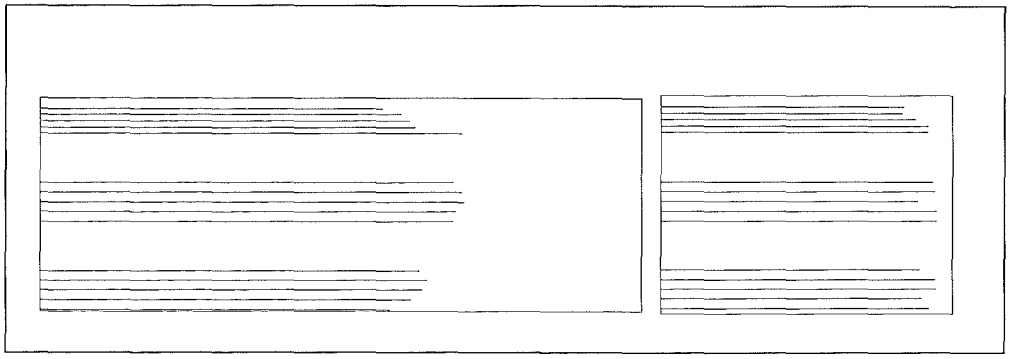


Figure 3.12: Selected peaks from the horizontal projection of potential staff systems.

Staff system 1:	[13, 13, 14, 14, 20, 20, 20, 20, 21, 21, 21, 21, 102, 105]
Staff system 2:	[12, 12, 14, 14, 20, 20, 20, 20, 21, 21, 21, 21, 103, 104]

Figure 3.13: The vertical separation between adjacent horizontal peaks, sorted into ascending order.

is at least twice as large as the previous item. The motivation for this search is that in the sorted list, all the distances caused by staff gaps will be grouped together, and all the distances between staves (staff jumps) will be grouped together; there will, therefore, be a large change of value where the two groups meet in the sorted list. A value of twice the previous value was chosen because it is unlikely that the separation between two staves within the same staff system would be less than twice the size of a staff gap. In fact, it would be hard for the eye to distinguish two staves that close together. Again, no example in the corpus contravened this threshold.

Due to noise and skew, the search for the gap between staves does not start at the beginning of the list. Unclean spikes in the horizontal projection may cause two (or more) local maximum turning points in one staff line peak. This results in a third distinct category of adjacent peak separations in the sorted list, and care must be taken to avoid selecting the jump from noise induced peaks to the distances between staff lines contained in the same staff.

Two solutions to this were investigated: an absolute minimum value, dependent on the scan resolution, that entries in the sorted list must exceed before they can be considered as a staff jump; and a relative offset, based on the length of the list, to determine a starting point for numbers to be considered as a staff jump. The motivation behind the first approach is to exclude the distance between the noise induced peaks from the calculation, which will always be a few pixels in separation. The motivation behind the second approach lies in the property that a staff has proportionally more staff lines in it than jumps from one staff to the next. Hence, in the case of staves consisting of five lines, the staff jump cannot occur in the first four-fifths of the sorted list. Variations on both methods are analysed in the results section below.

Once the minimum staff jump has been determined, the algorithm returns to the selected horizontal peaks (Figure 3.12) to establish the staff areas. Preliminary staff ar-

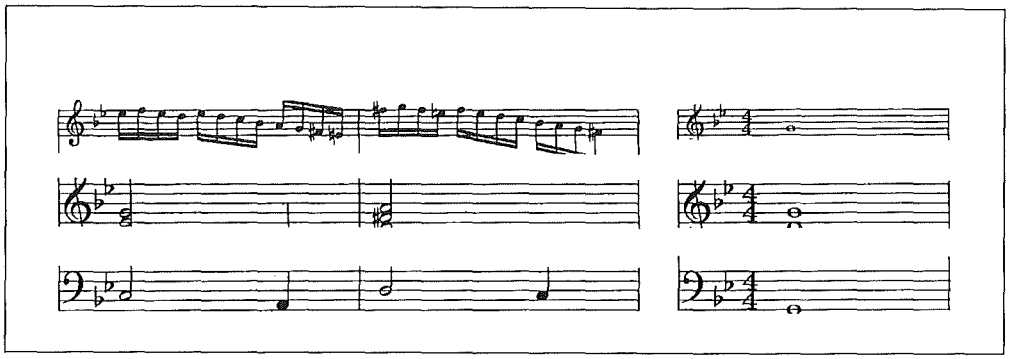


Figure 3.14: Potential staff areas.

eas are formed by grouping the horizontal peaks together, delimited by adjacent separations that equal or exceed the staff jump. To tolerate skew and noise, each staff area is enlarged by tracing the two peaks that mark the start and finish of the staff area back to the first minimum turning point below the cut-off threshold. This rectangular area is then further extended by 5% both above and below, to form the definitive staff area.

A situation that has not been discussed, so far, is one where a staff system consists of a single staff. In this situation no staff jump will be found. If such an event arises, the algorithm treats the entire shape as a staff area.

Figure 3.14 shows the regions detected by applying the algorithm described to the example excerpt of music. Notice how the algorithm detects staff areas, even if the relative size of the notation changes.

Evaluation

Choosing the staff jump based on an absolute setting was tested with the values 0.05 inch, 0.1 inch and 0.2 inch. For the relative position based approach, the starting point to search was tested at $\frac{3}{4}$ and $\frac{4}{5}$ of the way through the list. The result of applying these variations to the representational cross-section of the corpus is shown in Table 3.3 and Table 3.4 respectively. High values in the "number of staves missed" column indicate a poor algorithmic variation, whereas high values in the "number of extra staves" column are less important because extra staves are eliminated though subsequent processing.

An absolute value of 0.05 inch yields poor results because the value is too low, and consequently the search does not always clear the noise induced section of the sorted

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	5	39
Miniature score	10	0	7
Accompaniment	12	3	37
Monolinear (no chords)	15	1	5
Organ	12	8	45
Guitar	8	7	55
Piano	8	6	49
Popular piano	12	12	58
Hymn	8	8	39
Percussion	7	0	3
Computer generated	12	12	70
Handwritten	10	3	29
Sol-fa	8	0	16
Tablature	12	12	61
Square head notation	5	5	38
Total	154	82	551

(a)

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	15
Miniature score	10	0	7
Accompaniment	12	0	13
Monolinear (no chords)	15	0	1
Organ	12	0	5
Guitar	8	0	22
Piano	8	0	22
Popular piano	12	0	3
Hymn	8	0	3
Percussion	7	0	3
Computer generated	12	0	16
Handwritten	10	0	6
Sol-fa	8	0	15
Tablature	12	0	1
Square head notation	5	5	35
Total	154	5	167

(b)

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	2	15
Miniature score	10	0	7
Accompaniment	12	0	13
Monolinear (no chords)	15	0	1
Organ	12	0	5
Guitar	8	0	20
Piano	8	0	21
Popular piano	12	0	2
Hymn	8	0	3
Percussion	7	0	3
Computer generated	12	0	16
Handwritten	10	0	5
Sol-fa	8	0	13
Tablature	12	0	1
Square head notation	5	0	19
Total	154	2	144

(c)

Table 3.3: Accuracy of the potential staves algorithm using an absolute staff jump search when applied to the representational cross-section of the corpus (a) 0.05 inch (b) 0.1 inch (c) 0.2 inch.

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	15
Miniature score	10	0	7
Accompaniment	12	0	13
Monolinear (no chords)	15	0	1
Organ	12	0	6
Guitar	8	0	23
Piano	8	0	22
Popular piano	12	0	3
Hymn	8	0	3
Percussion	7	7	4
Computer generated	12	0	16
Handwritten	10	0	5
Sol-fa	8	0	12
Tablature	12	0	1
Square head notation	5	0	21
Total	154	7	152

(a)

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	15
Miniature score	10	0	7
Accompaniment	12	0	13
Monolinear (no chords)	15	0	1
Organ	12	0	6
Guitar	8	0	23
Piano	8	0	22
Popular piano	12	0	3
Hymn	8	0	3
Percussion	7	7	4
Computer generated	12	0	16
Handwritten	10	0	5
Sol-fa	8	0	12
Tablature	12	0	1
Square head notation	5	0	21
Total	154	7	152

(b)

Table 3.4: Accuracy of the potential staves algorithm using a relative staff jump search when applied to the representational cross-section of the corpus (a) $\frac{3}{4}$ into list (b) $\frac{4}{5}$ into list.

Staff Type	Recommended algorithm variation
CMN (only 5-line staves)	Relative setting $\frac{4}{5}$
CMN (including 1-line stave)	Absolute setting 0.1 inch
Guitar tablature	Relative setting $\frac{4}{5}$
Plainsong notation	Absolute setting 0.2 inch

Table 3.5: The recommended algorithm variations for the different staff types.

peaks. The values 0.1 inch and 0.2 inch fare better, with errors confined to a single image: the plainsong notation example in the case of the former, and the orchestrated score example in the case of the latter. The graphical properties of a staff differ between notations, and the ratio between the staff gap and staff line thickness for plainsong notation is greater than that for CMN. This is why the larger value identifies the true staff jump in the plainsong notation example, and the smaller value incorrectly finds the jump from noise induced peaks to staff gap, incorrectly classifying each staff line as a staff. The opposite situation occurs in the orchestrated score example.

Both relative based variations correctly identify all staves, except for the percussion example, where both make the same error. The sample percussion work includes two 5-line staves and five 1-line staves, so the assumption that the staff jump will not appear in the first $\frac{4}{5}$ (or $\frac{3}{4}$) of the list is no longer valid, and consequently no staff jump is found.

The tabulated results show that because the graphical properties of a staff differs from notation to notation, there is no single absolute setting, nor single relative setting that is guaranteed to work for all music. Table 3.5 shows the recommended algorithm variation for the different staff types. In an OMR application, these variations could be offered as options to be set accordingly by the end-user, based on the type of music being processed. Alternatively, all four variations could be tried, and a heuristic used to select the most likely scenario.

The tabulated results also show a high number of extra staves being detected by all the algorithms. This is because the objects tagged as staff systems are only *potential* staff systems, and potential staff areas are found in the erroneous staff systems. Subsequent processing will eliminate these erroneous areas.

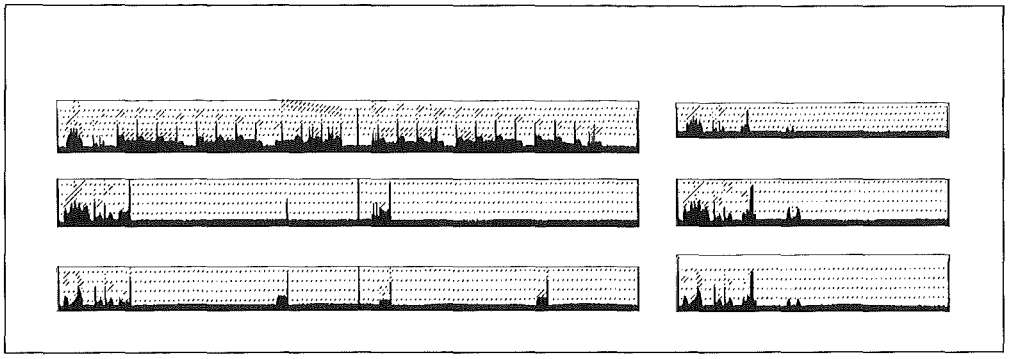


Figure 3.15: The vertical projection of each staff area.

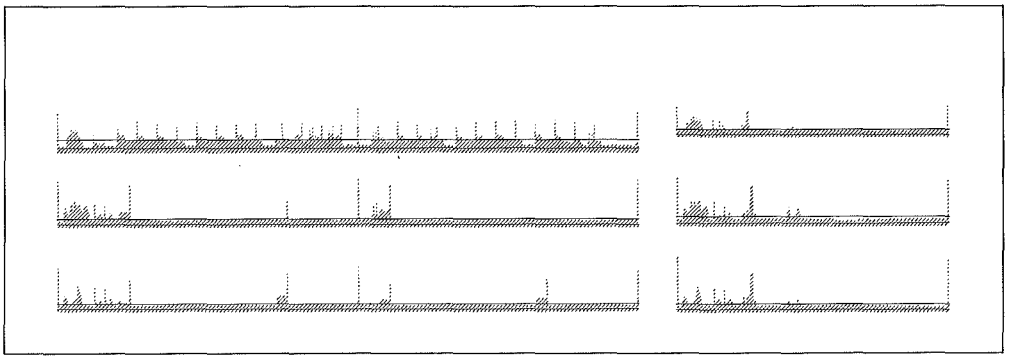


Figure 3.16: The vertical projection of each staff area, with upper and lower thresholds.

3.2.3 Potential staff segments

Now that the potential staff areas have been determined, each region identified is searched for "quiet" segments, where it is likely that only the staff lines appear. This is accomplished by taking a vertical projection of each staff area (Figure 3.15), and establishing where low trough-like segments in the projection are, aided by an upper and lower bound threshold (drawn as bold horizontal lines in Figure 3.16). The two thresholds are expressed as percentages of the average value for that particular vertical projection.

For each section of a vertical projection that falls between the two limits (Figure 3.17) the average value for the segment is calculated. The algorithm selects the segment with the minimum value as the best place to check for staff lines. By choosing the smallest average between the two limits, the algorithm favours lower, shorter segments, which is beneficial since this corresponds to segments of the staff area that are quieter and less affected by skew. Unfortunately, this selection process will discard many long low

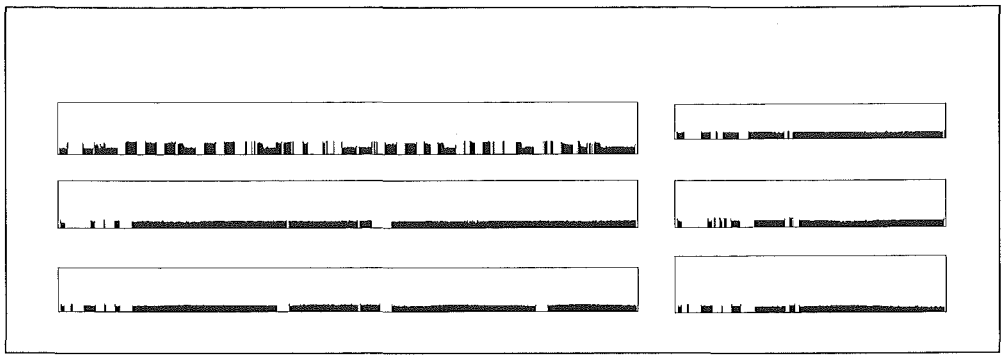


Figure 3.17: The low troughs that fall between the upper and lower thresholds.

troughs, common in bars that include notes or rests of a longer duration, which clearly satisfy the requirements of a "quiet" area. An additional step to the algorithm is to set a maximum length for low troughs, and subdivide any trough that exceeds this length into contiguous segments of the maximum length, plus a fractional residue. This is in fact better than considering the quiet region of the projection as one long section, since the segmented pieces are less affected by skew.

Once again the complications of noise intrude. Not only does the algorithm use a maximum limit to improve segment selection, but a minimum length is also used which a low trough must exceed before it is considered in the selection process.

Later on in the algorithm, the angle of skew the document was scanned at will be calculated from the position of the chosen staff segments. It is best, therefore, that the algorithm selects segments as far apart as possible, since this will help reduce the error bounds in the calculation. Similar to existing staff line detection algorithms, a staff segment is chosen (using the minimum average criteria) from the left-hand side of the page, and the right-hand side of the page (Figure 3.18).

Above we defined low trough-like segments to be those sections of the vertical projection that fell between two limits. This is an over simplification of the procedure used. A more intricate check is made using a form of hysteresis. Instead of an upper and lower threshold, there are in fact two boundaries for the lower threshold: a strict threshold and a more relaxed threshold. The start of a trough must fall between the upper threshold and the strict lower threshold. However, the trough may continue until it exceeds either the upper limit or the more relaxed lower limit. The motivation for this was once again practi-

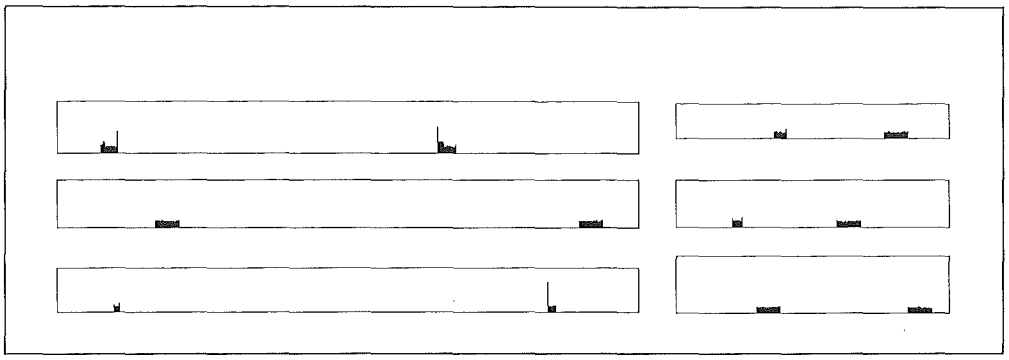


Figure 3.18: The low troughs chosen from the vertical projection: one from the left-hand side and the other from the right-hand side of the staff area.

	Upper threshold	Strict lower threshold	Relaxed lower threshold
Setting 1	125%	65%	45%
Setting 2	100%	33%	25%
Setting 3	75%	25%	20%

Table 3.6: Three different settings for the percentages of the vertical projection used to locate low troughs.

cal, permitting the thinning (and the possible brief disappearance) of individual staff lines.

Evaluation

As with other algorithms in this stage of the OMR system, there are choices for thresholds that must be set fairly arbitrarily. For this algorithm there are many degrees of freedom: the size of the region considered as the left-hand side, and the right-hand side of the staff area; the minimum and maximum length for a low trough; as well as the percentage levels used to determine the upper and lower bound thresholds when identifying troughs. Through empirical testing the algorithm was discovered to be stable with respect to all but the last category of parameter.

Table 3.6 shows the three different settings for the percentages of the vertical projection average used to investigate the sensitivity of low trough identification. Table 3.7 shows the results of processing the representational cross-section of the corpus with these values. The last column in the table records the number of pixels that differ between an "ideal" version of the staff lines in the image generated by hand, and the computer selected segments.

Category	Number of staves	Number of staves missed	Number of extra staves	Number of pixels that differ
Orchestrated score	15	0	2	944
Miniature score	10	0	2	912
Accompaniment	12	0	2	524
Monolinear (no chords)	15	0	0	1294
Organ	12	0	3	685
Guitar	8	1	15	995
Piano	8	2	19	3036
Popular piano	12	0	2	667
Hymn	8	0	2	494
Percussion	7	1	0	175
Computer generated	12	0	8	1190
Handwritten	10	0	3	2312
Sol-fa	8	0	4	206
Tablature	12	0	0	357
Square head notation	5	0	5	191
Total	154	4	67	13982

(a)

Category	Number of staves	Number of staves missed	Number of extra staves	Number of pixels that differ
Orchestrated score	15	0	0	823
Miniature score	10	0	0	679
Accompaniment	12	0	0	518
Monolinear (no chords)	15	0	0	1119
Organ	12	0	0	653
Guitar	8	0	0	1718
Piano	8	1	0	1404
Popular piano	12	0	0	407
Hymn	8	0	0	496
Percussion	7	0	0	137
Computer generated	12	0	0	697
Handwritten	10	0	0	1783
Sol-fa	8	0	0	250
Tablature	12	0	0	397
Square head notation	5	0	0	201
Total	154	1	0	11282

(b)

Category	Number of staves	Number of staves missed	Number of extra staves	Number of pixels that differ
Orchestrated score	15	0	3	480
Miniature score	10	0	0	562
Accompaniment	12	2	2	107
Monolinear (no chords)	15	0	0	507
Organ	12	3	2	420
Guitar	8	0	10	1160
Piano	8	1	16	1062
Popular piano	12	2	2	307
Hymn	8	0	1	281
Percussion	7	0	0	132
Computer generated	12	0	9	572
Handwritten	10	0	2	835
Sol-fa	8	0	2	139
Tablature	12	4	0	340
Square head notation	5	1	1	159
Total	154	13	50	7063

(c)

Table 3.7: Accuracy of the potential staff segments algorithm when applied to the representational cross-section of the corpus (a) Setting 1 (125%,65%,45%) (b) Setting 2 (100%,33%,25%) (c) Setting 3 (75%,25%,20%).

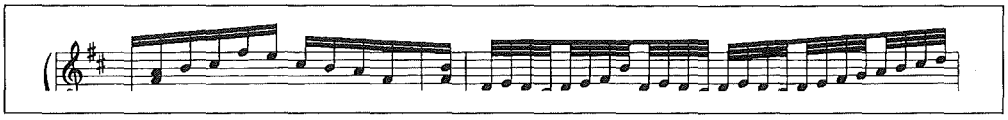


Figure 3.19: The staff from the sample piano piece in which the location of staff segments fails for all three variations of the algorithm.

Setting 1 fails to find staff segments in four of the staves tested. This is because the band of troughs sought is too high for certain images. Even the correctly chosen segments are not ideal since there is a tendency for segments to encroach into areas where there are musical features. This is why the total for the "number of pixels that differ" column is highest for Setting 1. In contrast, Setting 3 is too low for some images, and the algorithm fails to find segments for 13 staves spread across six images. As for Setting 1, there is also a high number of erroneous staves that pass the test for segments.

Setting 2 fares better, successfully eliminating all erroneous staves, and only failing to locate segments to one staff in the sample piece of piano music. In fact, all three algorithms fail on the same staff. Closer inspection of the piece reveals that the failed staff has a high percentage of beams that fall within the staff area (Figure 3.19). Consequently, the average computed for the vertical projection is further away from the "background" level caused by staff lines on their own. Widening the range between upper and lower thresholds to include staves like this, however, is inadvisable, as we have already seen that the higher and lower thresholds in Setting 1 and Setting 3 cause errors in other works.

A better solution is to introduce a second pass to the staff line detection algorithm. The second pass is identical to the algorithm presented, except that—now the average thickness of staff lines is known from the other staves successfully detected—a better estimate is available for the proportion of the vertical projection due to staff lines. The boundary limits computed as percentages of the *average value in the vertical projection*, are replaced with percentages of the *average contribution of detected staff lines per staff*. An application of this second pass algorithm successfully finds segments in the missing staff.

Even without this problem due to beams, this strategy of refinement is beneficial when processing a collection of pages. Instead of each page being processed in isolation, the average contribution of staff lines per staff can be used as the starting point for the detection of staff lines in subsequent pages. In fact, the idea can be taken further, with the

average staff height, staff gap, and staff line thickness being carried over, and exploited by the detection algorithm when applied to later pages.

Of course, if all the staves in a piece are like Figure 3.19 then the first pass of the algorithm will fail to find any staves, and consequently no average value for staff line thickness is available for the second pass. A further modification, therefore, is to alter the initial pass to detect such a failure, and to try the first pass algorithm again with different threshold values.

Given the above discussion of the tabulated results, we choose Setting 2 with a second pass of the algorithm that exploits the computed staff lines thickness from the first pass, as the algorithm for locating staff segments.

3.2.4 Potential staff lines

Figure 3.20 shows where the chosen staff segments occur in the example image. These regions must be investigated for evidence of staff lines. Like many preceding algorithms, this is accomplished using horizontal projections. For each region, a horizontal projection is taken (Figure 3.21), and a threshold based on the maximum value in the projection used to determine where the potential staff lines are. Empirical testing found thresholds in the range 25% to 75% reliable. A threshold of 50% was chosen.

3.2.5 Definite staff lines, staves, and staff systems

So far all the steps in this algorithm generate potential objects: potential staff systems, which leads on to potential staff areas, and then potential staff lines; objects that do not contain the next potential object type are discarded from the process. What remains at the end of this process should be staff lines, although a prudent step is to perform additional consistency checks, particularly when an extensible system is involved, since the set of musical shapes is not known ahead of time. This contrasts with many existing systems where a predefined limit of permissible shapes is often enforced. Such systems rely on simple tests, such as there must be five staff lines, to ensure that the located object is indeed a staff. More sophisticated checks are required by an extensible system.

Consistency checks for this project were carried out on the detected staff line thickness, and staff gap, per staff. For both items, the statistical measurements of range and variance:

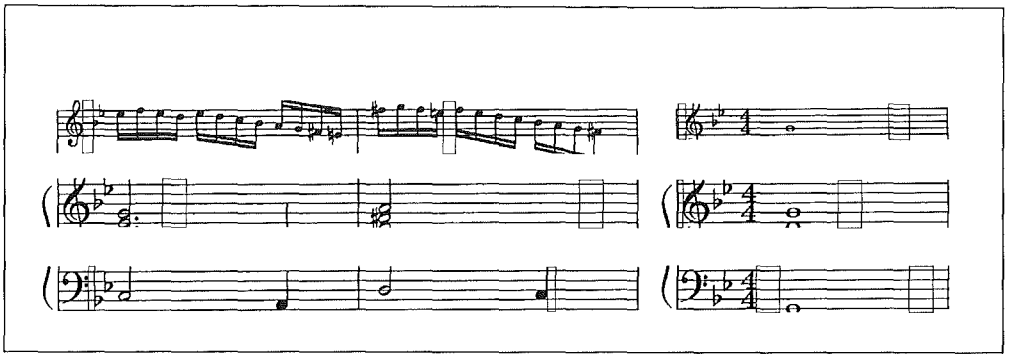


Figure 3.20: Where the selected staff segments occur in the detected staff areas.

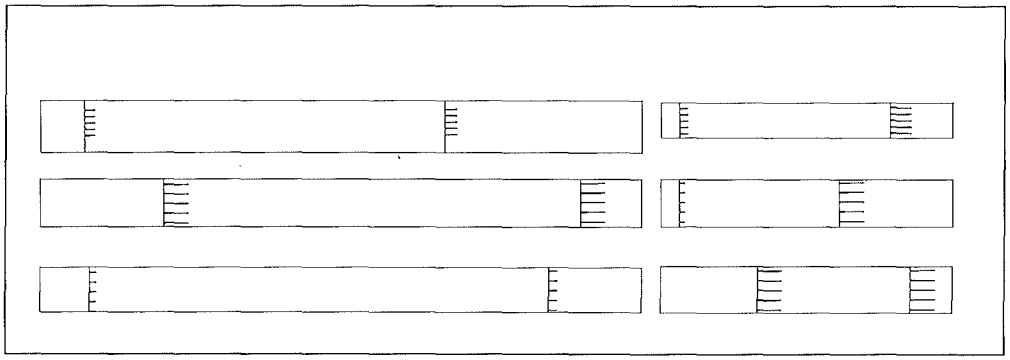


Figure 3.21: The horizontal projections of each staff segment.

$$range(X) = max(X) - min(X) \tag{3.1}$$

$$variance(X) = \frac{\sum_{x \in X}(x - mean(X))^2}{length(X)} \tag{3.2}$$

were computed. A true staff segment has staff lines and staff gaps that vary little in thickness. Once again thresholds had to be set to make such a classification. A variety of values were investigated to establish a reliable basis for choosing a set of values.

Evaluation

Table 3.8 shows three different settings for the consistency check, and Table 3.9 shows the result of applying these different settings to the representational cross-section of the corpus.

Setting 1 specifies constraints that are too tight for the sample piece of guitar tab-

	Staff line variance	Staff gap variance	Staff line range	Staff gap range
Setting 1	≤ 1.0	≤ 3.0	≤ 3	≤ 5
Setting 2	≤ 1.5	≤ 3.5	≤ 4	≤ 7
Setting 3	≤ 2.0	≤ 4.5	≤ 4	≤ 7

Table 3.8: Settings used for staff consistency check.

lature, and two of the 6-line staves in this image fail the test because the calculated gap variance and gap range both exceed the set thresholds. Subsequent settings are more tolerant of deviations from the norm, and by Setting 3 all staves in all images pass. Setting 3, therefore, is chosen as a reliable consistency check. Relaxing the thresholds further increases the chance of erroneous staves being included.

3.3 The final algorithm

Table 3.10 summarises the algorithm variations found to be the most reliable for each step in the new staff detection algorithm. The result of applying this algorithm to the opening page of every entry in the corpus is tabulated in Table 3.11.

The accuracy rate of 97% for CMN shows a high degree of reliability. The main reason for 3% failure was due to the incorrect selection of staff jumps. In the images where this occurred, the incorrectly processed staves included so many notes heads and other musical objects drawn either above or below the staff, that the simple search used to find the staff jump to the next staff is obscured. Two erroneous staves were also detected, but these were both a consequence of incorrect staff jumps. Reliability of the staff detection algorithm could be increased by replacing the staff jump selection process with a more sophisticated version.

The staff detection algorithm assumes that all staff lines in a staff are vertically connected, but with plainsong notation this is not always true. In Figure 3.22 both the top and bottom staff lines never come into contact with the central part of the staff. This is why the accuracy rate for plainsong notation is markedly lower, at 73%. For the 38 staves where the staff lines within a staff are connected, the algorithm correctly identifies them all with no erroneous staves being included.

One remedy to this staff line separation problem is to incorporate a post process-

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	0
Miniature score	10	0	0
Accompaniment	12	0	0
Monolinear (no chords)	15	0	0
Organ	12	0	0
Guitar	8	0	0
Piano	8	0	0
Popular piano	12	0	0
Hymn	8	0	0
Percussion	7	0	0
Computer generated	12	0	0
Handwritten	10	0	0
Sol-fa	8	0	0
Tablature	12	2	0
Square head notation	5	0	0
Total	154	2	0

(a)

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	0
Miniature score	10	0	0
Accompaniment	12	0	0
Monolinear (no chords)	15	0	0
Organ	12	0	0
Guitar	8	0	0
Piano	8	0	0
Popular piano	12	0	0
Hymn	8	0	0
Percussion	7	0	0
Computer generated	12	0	0
Handwritten	10	0	0
Sol-fa	8	0	0
Tablature	12	1	0
Square head notation	5	0	0
Total	154	1	0

(b)

Category	Number of staves	Number of staves missed	Number of extra staves
Orchestrated score	15	0	0
Miniature score	10	0	0
Accompaniment	12	0	0
Monolinear (no chords)	15	0	0
Organ	12	0	0
Guitar	8	0	0
Piano	8	0	0
Popular piano	12	0	0
Hymn	8	0	0
Percussion	7	0	0
Computer generated	12	0	0
Handwritten	10	0	0
Sol-fa	8	0	0
Tablature	12	0	0
Square head notation	5	0	0
Total	154	0	0

(c)

Table 3.9: Accuracy of the staff segment consistency check when applied to the representational cross-section of the corpus (a) Setting 1 (1.0,3.0,3,5) (b) Setting 2 (1.5,3.5,4,7) (c) Setting 3 (2.0,4.5,4,7).

Potential staff systems	8-pixel (byte) flood-fill	
Potential staff areas	CMN (only 5-line staves)	Relative setting $\frac{4}{5}$
	CMN (including 1-line stave)	Absolute setting 0.1 inch
	Guitar tablature	Relative setting $\frac{4}{5}$
	Plainsong notation	Absolute setting 0.2 inch
Potential staff segments	Upper threshold	100%
	Strict lower threshold	33%
	Relaxed lower threshold	25%
Potential staff lines	50% of maximum peak	
Consistency check	Staff line variance threshold	2.0
	Staff gap variance threshold	4.5
	Staff line range threshold	4
	Staff gap range threshold	7

Table 3.10: The final staff detection algorithm.

Type of notation	Number of works	Number of staves	Number of staves correctly identified	Accuracy
Common music notation	59	529	512	97%
Plainsong notation	7	52	38	73%

Table 3.11: Accuracy results from processing the opening page from each corpus entry.

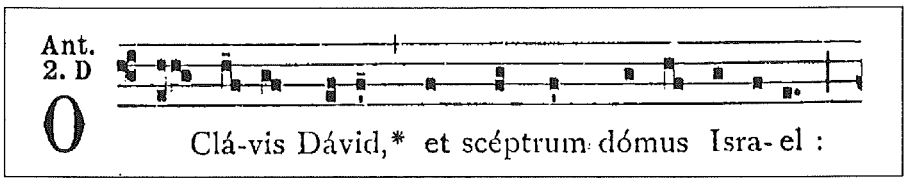


Figure 3.22: In plainsong notation staff lines in a staff are not necessary connected.

ing phase that merges two potential "staff systems" together if the vertical separation between them is less than or equal to the detected staff gap.

This customised solution again highlights the difficulty in developing a single, all encompassing algorithm to detect staves from a variety of notations. The trend seen in this chapter is the development of a basic methodology that includes specific variations for particular forms of notation made available as options in the application.

Chapter 4

Musical object location

With the staff lines identified, and consequentially the staff heights inferred, the OMR system has extracted some information about the scanned image. However, additional image processing is still required before the musical shapes can be identified graphically. The key task is the location of the musical shapes, many of which are superimposed on staves.

In this chapter, the task of musical object location is explored. First, existing work is reviewed, from which the competing methodologies of *removing staff lines* and *crossing over staff lines* to isolate musical shapes are identified as suitable candidates given the project's aims of extensibility and complete graphical reconstruction. Variations on the two approaches are described and evaluated.

"Bach Chorale No. 367," shown in Figure 4.1, will be used as a running example throughout this chapter. In detecting the staves (Chapter 3) preliminary processing of the image has already been performed: an OCR algorithm[1] has detected and removed text and text-like objects (Figure 4.2); and the staff detection algorithm has established the size of the music notation (Table 4.1). Figure 4.3, therefore, is the starting point for this stage in an OMR system.

4.1 Isolated musical object location

A large quantity of musical shapes that appear on a page are printed in an "uncluttered" manner. That is to say, many glyphs do not overlap or touch any other glyph. Exam-

[1]The OCR software used was a modified version of a program initially written for internal use by the Department of Computer Science, Trinity College, Dublin, Ireland.

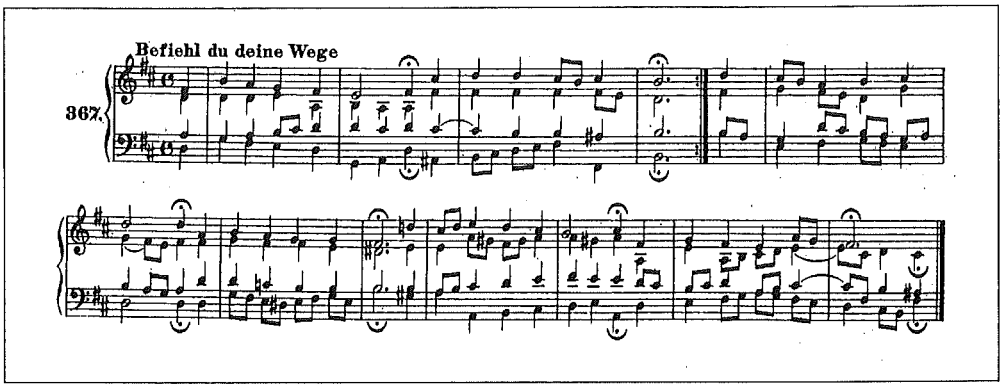


Figure 4.1: The example piece of music used throughout the chapter. The piece is skew and warps upwards at the left-hand side.

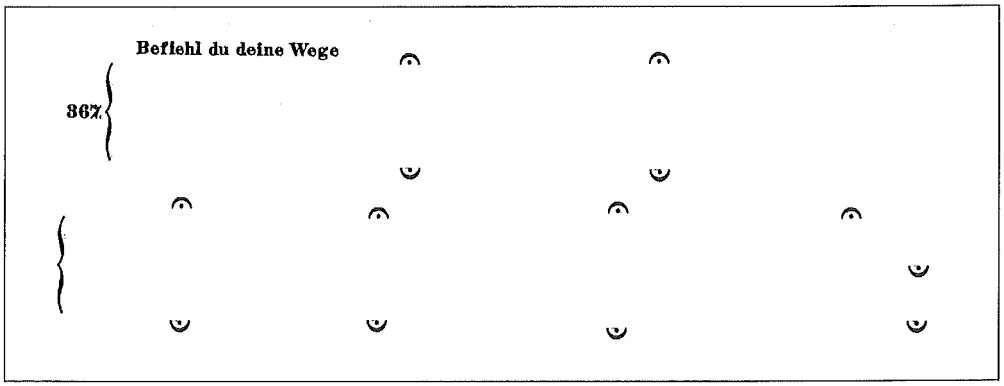


Figure 4.2: Text and text-like objects located in the example piece of music using a template based OCR system, where a library template can include disjoint shapes, as in the case of the fermata.

ples of this can be seen in Figure 4.4, where lyrics, guitar fingering, and musical directions (such as "TO CODA ⊕") are all printed clearly and separately. Such shapes are easy to isolate. Although it is not as straightforward as extracting the bounding box to each shape—since a bounding box may inadvertently crop nearby shapes—the repetitive flood-filling and removal of shapes will correctly isolate these wholly connected marks. This process has already been described earlier (Section 2.3, page 22) where it was used to solve a similar shape extraction problem.

Average staff line thickness	3 pixels
Average staff gap height	13 pixels
Average staff height	66 pixels
Average angle of skew	-0.236°

Table 4.1: Known music notation information for the example piece (scan resolution 300 dpi).

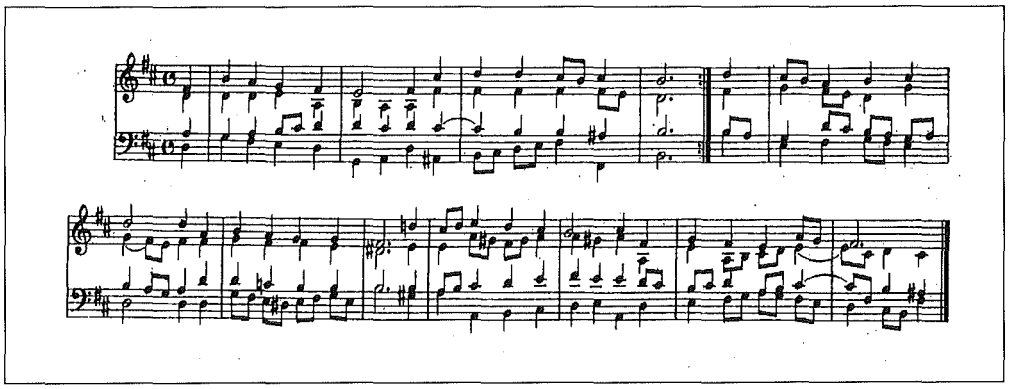


Figure 4.3: The starting point for the object location stage of an OMR system.

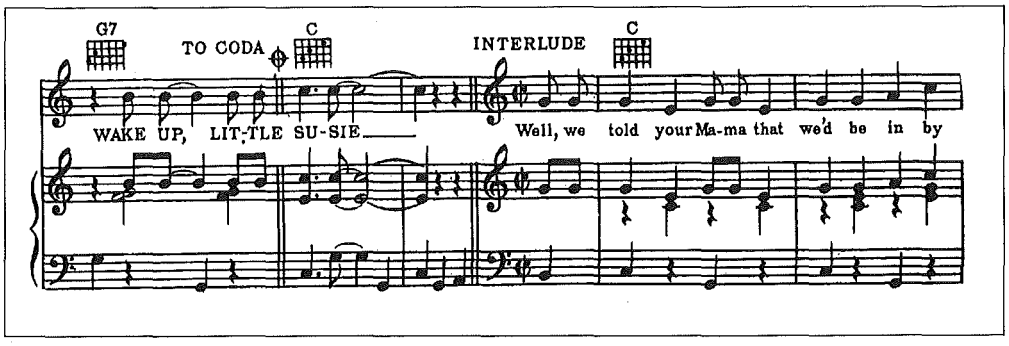


Figure 4.4: Many musical objects are printed separately.

4.2 Superimposed musical object location

This leaves the musical shapes that are printed on the staves. Here we are confronted by the issue of *superimposition*, the attribute that distinguishes music from many other forms of structured document. To process the music, the computer must separate the musical shapes from the staves. Fortunately this task is aided by the regularity and predictability of the staff lines.

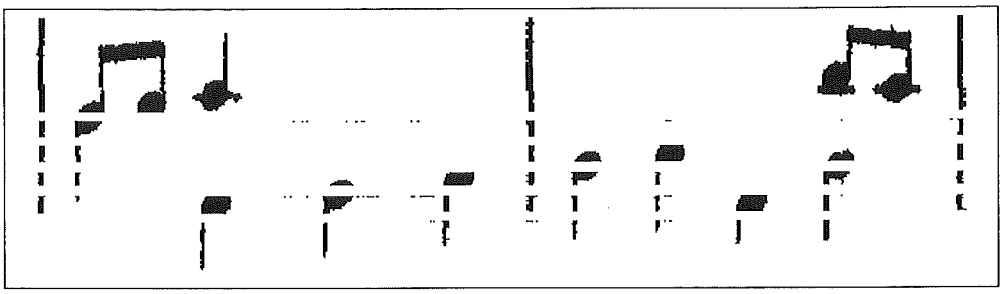


Figure 4.5: An excerpt from the example piece of music (bass line, bars 9 and 10 of Figure 4.1) with staff lines removed using the algorithm described by Prerau.

Existing approaches to locating superimposed objects fall into three categories: location by *removing* the staff lines, location by *bumping* into the shapes between the staff lines, and location using a hybrid model that eliminates staff lines by *crossing* over the staff lines to get from one shape that lies between two staff lines, to another adjoining shape.

4.2.1 Removing staff lines

Prerau [Pre70] removed staff lines completely, without regard for other musical features, thus dissecting all shapes that overlapped the staff. The effect of doing this is shown in Figure 4.5. Assembling the remaining pieces was a substantial part of his work.

A more popular technique devised after Prerau, which keeps most musical features whole, was first described by Clarke *et al.* [CBT88a]. The algorithm is based on the notion of a vertical *slither*—a section of the image that is one pixel wide. To remove staff lines, this algorithm traverses the length of each staff line (say from left to right), using the template shown in Figure 4.6 to determine if there is an object superimposed on the staff line at any given point; for a vertical slither of staff line starting at pixel A, if pixel B is black and at least one of the pixels X, Y or Z is black, then this is deemed sufficient evidence of an object, and the slither of staff line remains. A mirror image of the template is used for the bottom half of the staff line. The result of applying this algorithm to the same excerpt of music as before, is shown in Figure 4.7.

Even with this careful staff line removal algorithm, parts of the staff line remain, and some objects still become fragmented. Due to the high level of noise in the example scanned image, the staff line thickness varies considerably. Back in Figure 4.5, places where the staff line thickness exceeds the expected value appear as spurious debris in the dissected

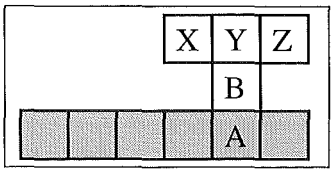


Figure 4.6: The template proposed by Clarke *et al.* for removing staff lines. The gray pixels represent a staff line.

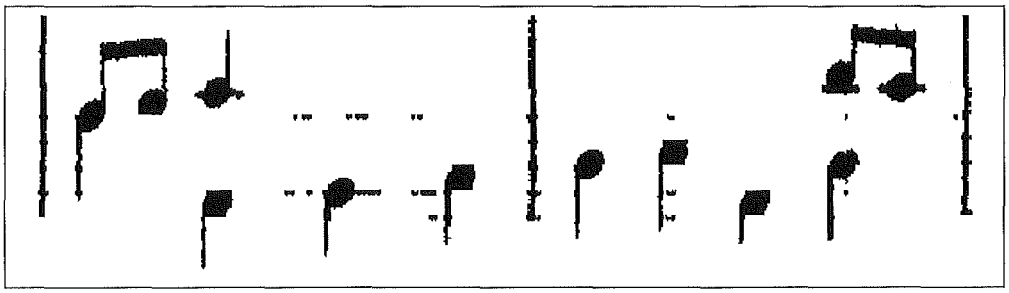


Figure 4.7: An excerpt from the example piece of music (bass line of bars 9 and 10) with staff lines removed using the algorithm described by Clarke *et al.*

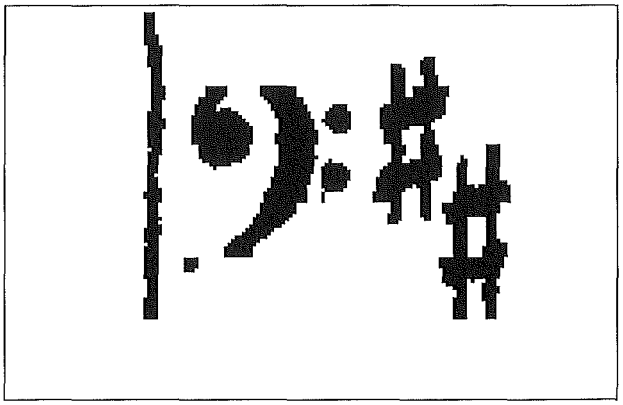


Figure 4.8: Even when care is taken, fragmentation can still occur.

image. In the more careful staff line removal algorithm, these same places falsely provide evidence that a musical object exists at this point. In contrast, fragmentation arises when a thin part of a musical feature, typically a line, blends tangentially into a staff line. This is particularly common for minim note heads and the bass clef. For example, Figure 4.8 shows the effect of this algorithm on the lower bass clef in the sample music. Not only is the top of the bass clef fragmented, but also the part of the clef that crosses the B line.

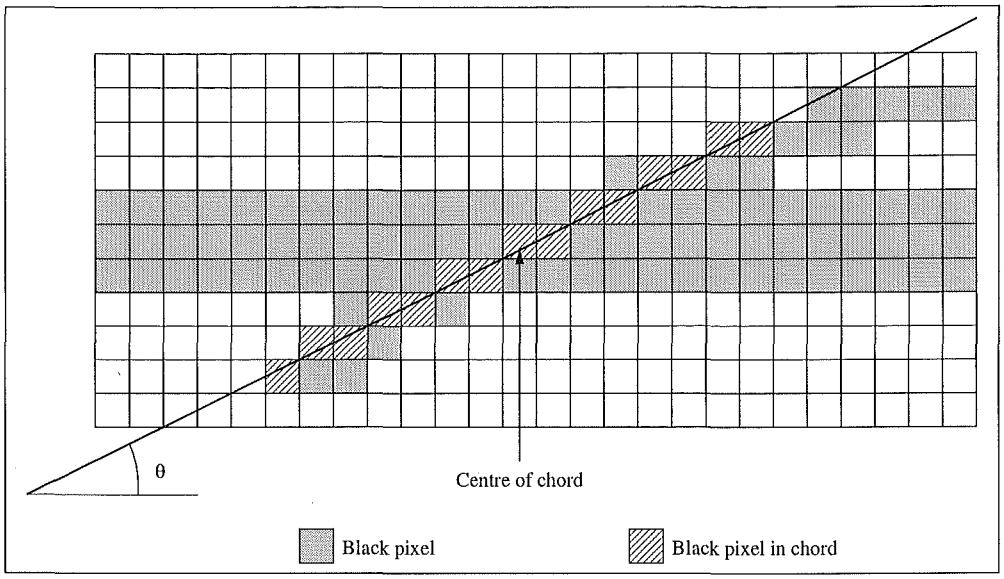


Figure 4.9: An example (mathematical) chord.

An algorithm described by Martin and Bellissant [MB91] addresses this deficiency. It is similar to that described by Clarke *et al.* except that a more intricate computation is used to decide when to remove a slither of staff line.

Instead of using a template to decide if an object passes through a particular point on a staff line, a set of *chords* are taken. Here the term chord is used in its mathematical sense, to mean a straight line joining the ends of an arc. The terminology is slightly abusive, since we are not strictly dealing with circles, but with the irregular shapes that cross staff lines. Nevertheless, the intention is the same, since the algorithm computes the shortest distance between two points on the perimeter of the shape. An example of this mathematical type of chord is shown in Figure 4.9.

In the algorithm, chords are computed over the range [−90°,+90°], and assembled into a histogram based on angle. A histogram with one distinct peak lying at approximately 0° is caused by a point on the staff line that has no object passing through it, and therefore, can be safely removed. A histogram with two or more distinct peaks is the result of an object passing through, or nearby, that point on the staff line, and is consequently left in the image.

Due to noise, histogram peaks are not smooth, and consequently they are not sought directly. Instead, a neural network [RHW86] is used, with hidden units that are con-

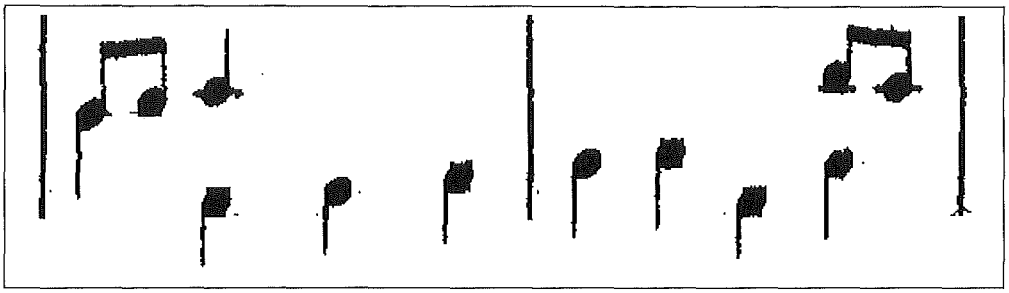


Figure 4.10: An excerpt from the example piece of music with staff lines removed using the algorithm described by Martin and Bellissant.

stricted in their input values to adjacent angles[2] [Sol89].

For practical purposes, the chord technique uses the Bresenham integer line algorithm [FvDFH90] to trace the chords, and is restricted to a 50×50 box of pixels. Also, due to the increased computational cost of this new test for an object passing through a staff line, the test is only applied to the centre of the staff line, rather than the top and bottom.

With hind-sight, we can now see that the template method used by Clarke *et al.* is a simplified version of this chord-based algorithm. Instead of checking out all possible lines, only three chords are checked top and bottom. From Figure 4.6, the chord lines at the top are BX, BY, and BZ. The lines themselves are a mere two pixels long. It is not surprising, therefore, that the simpler method does not correctly process so many types of superimposed shapes.

The result of applying this chord based algorithm to the same excerpt of music used to demonstrate the other removal algorithms is shown in Figure 4.10. The algorithm does indeed reduce fragmentation. In Figure 4.11 the same enlargement that revealed a fragmented bass clef when using the algorithm by Clarke *et al.*, is shown after an application of the chord based algorithm. This time the top part of the bass clef remains connected.

Martin and Bellissant reported that although their algorithm does reduce cases of fragmentation, it does not correctly preserve all cases. Additionally, some objects that should remain separate (and would have remained separate under the basic algorithm) become joined. An example of this can be seen in Figure 4.11, where the two sharps have become joined.

[2]The range of angles is considered cyclic. In other words, $-90°$ wraps around to $+90°$.

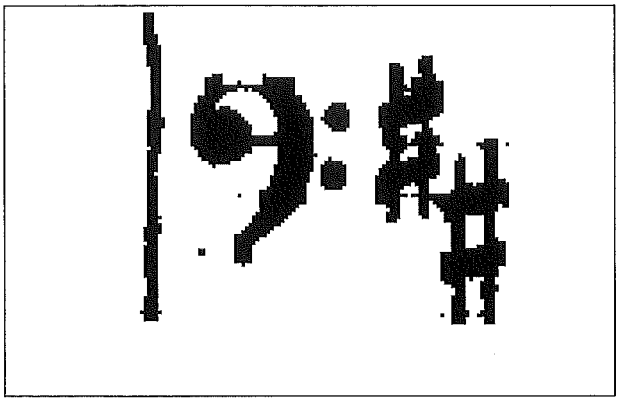


Figure 4.11: Even when great care is taken to remove staff lines, mistakes still occur.

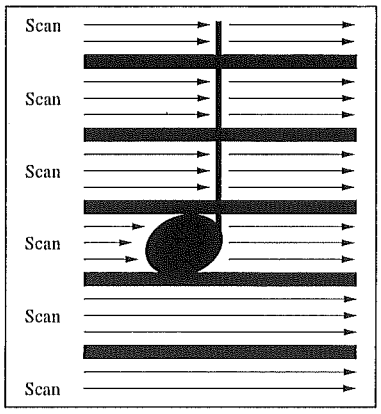
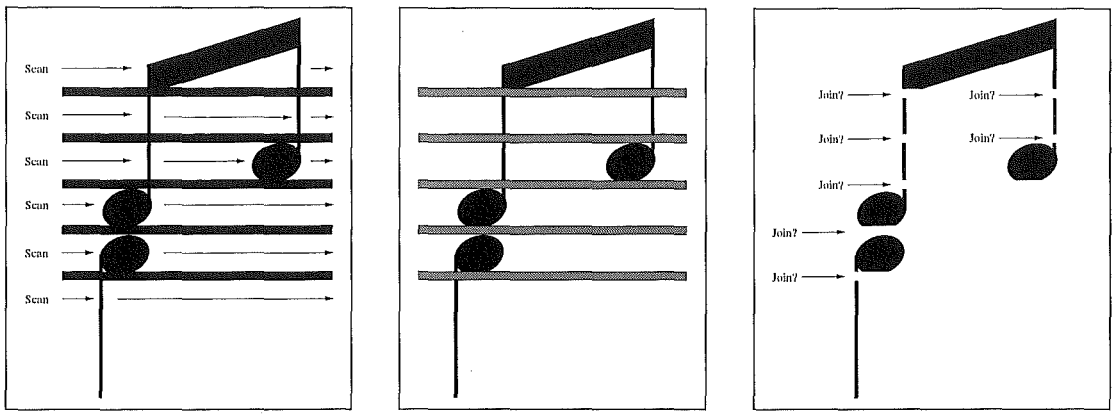


Figure 4.12: Bumping into shapes between the staff lines.

4.2.2 Bumping into shapes between staff lines

Rather than removing the staff lines to locate objects, the algorithm used in the Wabot-2 project [MHS+85] ignores the staff lines. The space between staff lines is systematically checked for the presence of objects, as is shown in Figure 4.12. When a shape is "bumped" into, contextual information is used to aid classification. For instance, after travelling a small distance to skip over a staff system's initial bar line, a clef must be the first shape encountered on a staff, optionally followed by a key signature.

A primary function in this style of algorithm is the detection of note heads. For each place where a note head has been located, a check is then made for a stem, and if that is successful, for beams, and tails. Also, the area to the right of the note head is checked for duration dots, and the area to the left is checked for accidentals.



(a) (b) (c)

Figure 4.13: Separating the combined musical object location and musical feature classification strategy leads to a situation similar to Prerau's simplistic staff line removal algorithm (a) scanning between the staff lines (b) the shapes bumped into shown in black (c) the numerous isolated shapes that need to be classified and assembled.

The Wabot-2 project was the first significant work that chose to ignore staff lines, and has had much influence over later projects that chose the same route [LC85, Gla89, IIHO92, MRHS95]. Good results have been achieved using this approach, but when considering the requirements of this project, the method, unfortunately, is inappropriate. In this method, the stages of musical object location and musical feature classification are combined. Both components feed off each other, forming a cyclic communication. This violates our restriction in the OMR framework to one-way communication links between stages (page 33, Figure 2.15) which was made to control the development of the project.

It is possible to separate the two. First, all the shapes between the staff lines are located, and stored in a list; then the musical feature classification algorithm is applied to the list. In doing this, however, we have weakened the approach by losing the interaction between locating the shapes between the staff lines, and musical feature classification. With interaction, the classification just made can be used to direct where to search for the next shape, and properties about the next object found can be used to alter how the classification algorithm is invoked. Without interaction, we have reduced the problem to a form similar to Prerau's simplistic staff line removal algorithm, where we have many fragmented shapes that need to be classified and assembled (Figure 4.13).

Fortunately, there is a variation on the bumping into shapes method that fits our

OMR framework, which is explained next.

4.2.3 Crossing over staff lines

In a prototype OMR system devised by the author, superimposed objects were located by crossing over staff lines [Bai91].[3] The algorithm combines the ideas of bumping into shapes, and eliminating the staff lines, without violating our restriction to a one-way communication framework for OMR. The algorithm starts in the same way as the bumping into shapes strategy, by tracking between staff lines for shapes. When one is found, a modified flood-fill algorithm that crosses over staff lines is used to locate the entire object, rather than the shape that falls between the two staff lines. This is where the algorithm is similar to the removal method, since the decision to cross a staff line is a reshaped form of the heuristic used to decide if a slither of staff line should be removed.

In the prototype system, the crossing of staff lines is achieved by introducing two extra rules into the flood-fill algorithm that control which points are pushed onto the stack. If the current pixel is one vertical pixel away from a staff line:

- neighbouring pixels to the left and right are *not* pushed onto the stack, and
- the pixel on the opposite side of the staff line is pushed onto the stack if there is sufficient evidence of the object continuing on the other side.

The rule devised by Clarke *et al.* [CBT88a] to decide if part of a staff line should be removed is used to determine if the object continues on the opposite side of the staff line. This is illustrated in Figure 4.14, where the gray pixels represent a staff line. If A is the current pixel and both W and D are black, then: D would be pushed onto the stack as normal; B and C would not be pushed on; and W would only be pushed on if X, Y or Z were black.

Unfortunately, ignoring staff lines in this manner leads to fragmented objects. The circumstances are identical to those that plague the removal methods, namely when parts of musical objects blend into the staff lines.

[3] In the author's Honours report [Bai91] the term "jumping" was used. The term "crossing" is now preferred.

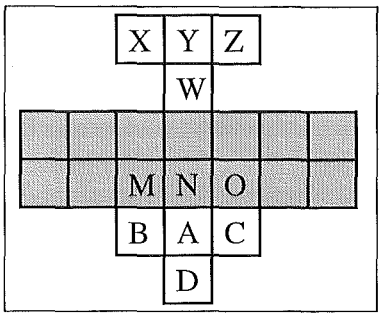


Figure 4.14: The template used to cross over staff lines. The gray pixels represent a staff line.

4.2.4 Removing versus crossing staff lines

Both a staff line removal strategy and a staff line crossing strategy are suitable for this project. Since the quality of object location is identical, the only question is whether there is any marked difference in speed of the two approaches.

Both algorithms, at some point, have to traverse the areas delimited by the staff systems, checking for black pixels and performing some version of a flood-fill algorithm each time one is found—so there is little difference there. A difference, however, does occur in the number of times they must determine if a musical object is superimposed on a staff line. In the case of a removal algorithm, this must be performed over the entire length of every staff line. In the case of crossing over staff lines, this need only be performed where parts of objects are superimposed with staff lines. A caveat to this, though, is that to cross staff lines, the basic flood-fill algorithm has been modified, and this itself carries an additional cost. Admittedly, the change is only slight, adding a check of atomic complexity that determines if the current pixel in the flood-fill algorithm is immediately above or below a staff line, but it is embedded at the heart of the function.

The crucial question is therefore, does the searching for black pixels and subsequent flood-filling dominate the cost of the superimposed musical object location algorithm, or the total cost of deciding which parts of objects are superimposed on staff lines? This in turn, depends on the heuristic being used to "solve" the superimposition problem. Consequently, no theoretical answer can be given. Experimentation is needed to clarify the situation, the results of which appear in Section 4.4.3.

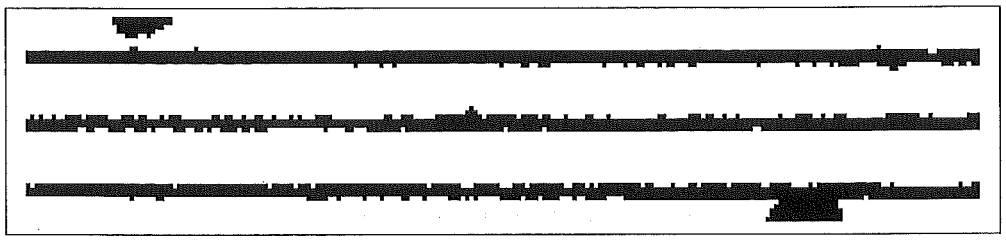


Figure 4.15: In a scanned image, the thickness of a staff lines can vary by a pixel or so.

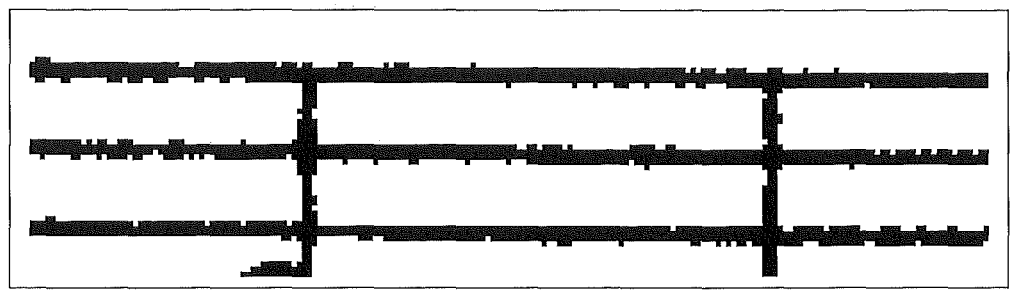
4.3 Refinements of existing techniques

Both removal and staff crossing algorithms have been used successfully in OMR systems. Their use to date, though, has been ad hoc, with implemented systems committed to a particular strategy arbitrarily. Here we aim to not only refine the reported algorithms, but also compare the performance of rival algorithms, so that a more informed decision can be made as to where and when a particular algorithm should be used.

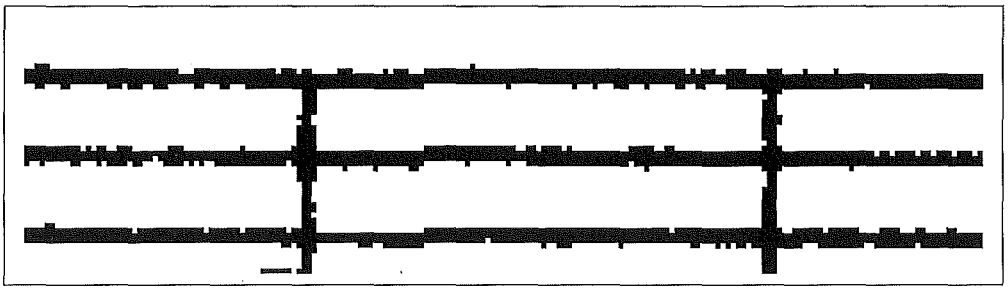
In the literature, algorithms that remove staff lines are described using diagrams that show the staff lines as level (Figures 4.6 and 4.14 are no exception). This implies that an image has been corrected for skew before this operation is carried out. There is, of course, no reason why the staff lines must be level. By incorporating the gradient into the algorithm it is easy to follow an angled line using, say, Bresenham's line algorithm [FvDFH90].

A second refinement concerns tolerance. The algorithms described in the literature are designed to tolerate small amounts of irregularities in the image, since a staff line does not have a constant thickness over the whole line—deviating up or down by a pixel or so. This phenomenon is clear in Figure 4.15. Such fluctuations are accommodated in the methods discussed, but the manner in which the algorithms are described indicates a reaction to the symptoms, rather than an analysis of the cause. Understanding the cause of the irregularities will improve our targeting of the algorithm. There are two primary sources to the fluctuations: the quality of the original, and aliasing effects in the digitization process.

Aliasing is a sampling error that occurs when a function of a continuous variable that contains sharp changes in intensity is approximated by discrete measurements. In OMR work the effect of aliasing is felt in many ways. Not only does it produce the fluctuation in staff line thickness, but it also causes a stepping phenomenon in staff lines that are scanned skew. Here aliasing strikes twice: first when trying to approximate the orig-



(a)



(b)

Figure 4.16: Correcting an image for skew causes aliasing effects (a) an excerpt of the example piece of music before rotation (b) the same excerpt after it has been corrected for skew.

inal smooth lines during scanning; and a second time, when using the mathematical formula for the detected staff lines in reverse, to predict where the staff lines should be in the scanned image. Unfortunately there is no guarantee that the stepping caused by the two forms of aliasing will coincide.

Another area where aliasing is implicated is during image enhancement. Later on in Chapter 5, when we investigate methods that correct an image for skew, we will see a low frequency oscillation effect in the staff lines. Figure 4.16a shows an excerpt of the example piece before processing, where the staff lines slope down to the right. Figure 4.16b shows the excerpt after correction for skew, where the staff lines are now level, but oscillate between two positions.

4.3.1 Removal

If the aliasing effects of a staff line thickness momentarily increasing by one pixel *and* the stepping phenomenon raising the position of the line by one pixel coincide, then pixels from

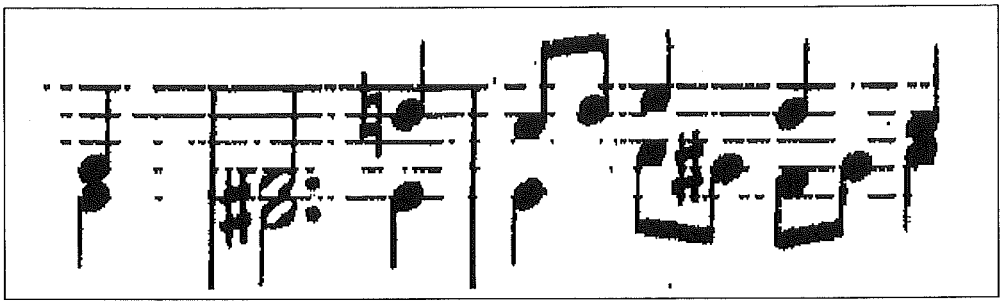


Figure 4.17: Aliasing effects can cause poor staff line removal when using a direct implementation of the algorithm described by Clarke *et al.*

the staff line in the scanned image would be placed two pixels higher than their anticipated location. It is possible, therefore, that the template proposed by Clarke *et al.* would accept these stray sections of staff line as sufficient evidence of a musical object being superimposed on a staff line at that point. Figure 4.17 shows a different part of the example image, where the application of the algorithm yields poor staff line removal, for this reason.

Given the above discussion, it is not surprising that a direct implementation of the algorithm is too rigid, and the same is true for all the staff line removal methods discussed so far that are based on slithers. What is needed is a flexible way to decide where a slither of detected staff line really is. This ability becomes increasingly important if deformities such as bowing staff lines are to be tolerated, since the potential deviation is no longer constrained to one or two pixels. Accommodating bowing staff lines and aliasing motivated the development of two strategies: *wobble* and *track*.

Figure 4.18 shows an overview of the *wobble* process. Working from left to right, the algorithm attempts to follow the staff line by taking slithers that are one pixel wide. Based on the previous slither of staff line detected, the next slither is searched for in a nearby vertical location. The slither found can be slightly higher or lower than the previous slither, hence the ability to follow a staff line that wobbles (or bows or bends).

Figure 4.19 shows an overview of the *track* process. Working from left to right, the algorithm uses the predicted location for the staff line as a starting point. If a slither of black does not pass through that point then the vertical areas above and below are searched for suitable slithers, and the one closest to the expected position is chosen.

The procedure for one slither in the *wobble* process is outlined in Figure 4.20. The starting point is the *top* and *bottom* of the previous staff line slither, which is then widened

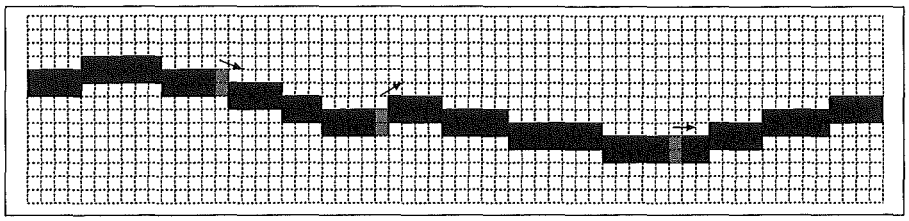


Figure 4.18: Overview of the *wobble* process.

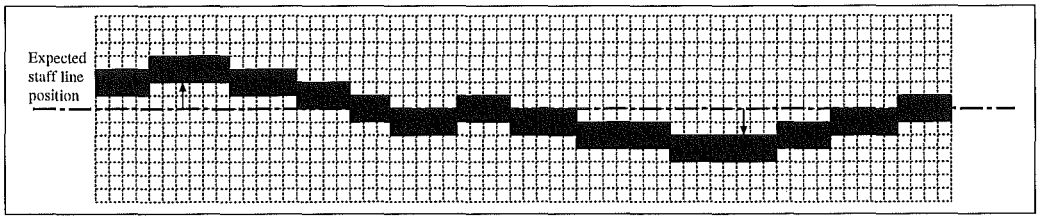


Figure 4.19: Overview of the *track* process.

by one pixel both above and below (Figure 4.21); this is done to allow the next slither of staff line to deviate away from the previous position. Next, all the slithers that at least start between the new values of *top* and *bottom* are found, and the centre of the slither that is closest to the centre of the previous slither is chosen. Since it is possible that the slither found is due to an object passing through the staff line at that point, the final step is to check that the chosen slither is short enough to be only a staff line. If this is not the case, then the previous values for *top* and *bottom* are used as the extremities of the new slither.

The procedure for one slither in the *track* process is outlined in Figure 4.22. The starting point is the predicted location of the centre of the staff line for the given column of the image we are working with. If this pixel is black, then we have either struck the staff line or an object passing through the staff line at this point. The slither that includes this pixel is determined and its length checked. As with the *wobble* process, if the slither is too long then the previous values of *top* and *bottom* are used. If the predicted pixel is white, then the staff line has deviated from its expected position (or else is missing, perhaps due to a poor quality original). In this case, the closest binding slithers above and below are found that start at a distance no more than "twice the average thickness of the staff lines" away from the previous slither. If no slither is found, then the data for that adjacent area to the staff line is tagged as invalid. For only the valid slithers, the closest

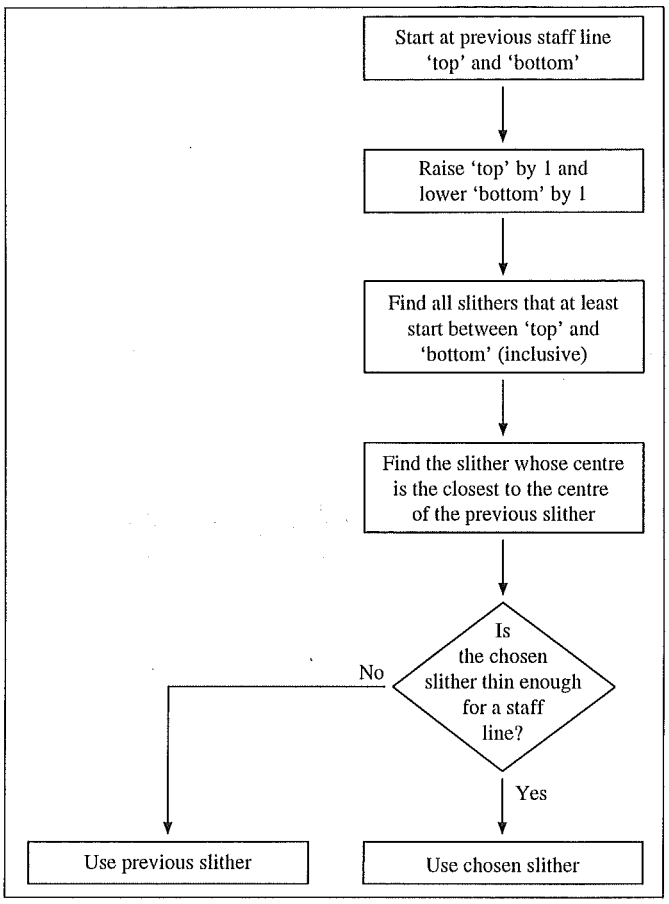


Figure 4.20: The procedure for choosing one slither in the *wobble* process.

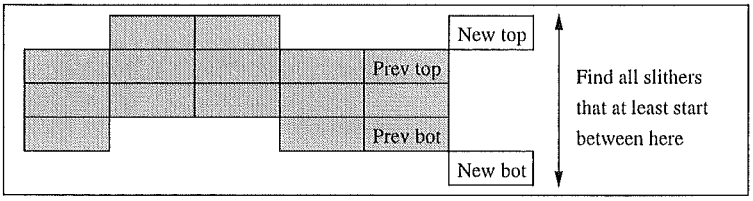


Figure 4.21: How the limits for potential new slithers are based on the previous slither of staff line.

one to the previous slither is chosen, and its length is checked against the staff line thickness. If, during this selection process, all the possible choices are eliminated, then the extremities of the previous slither are used to form the new slither.

To process the isolated musical features, it is not necessary for a removal based algorithm to retain an image of the removed staff lines. However, if—later on in the OMR

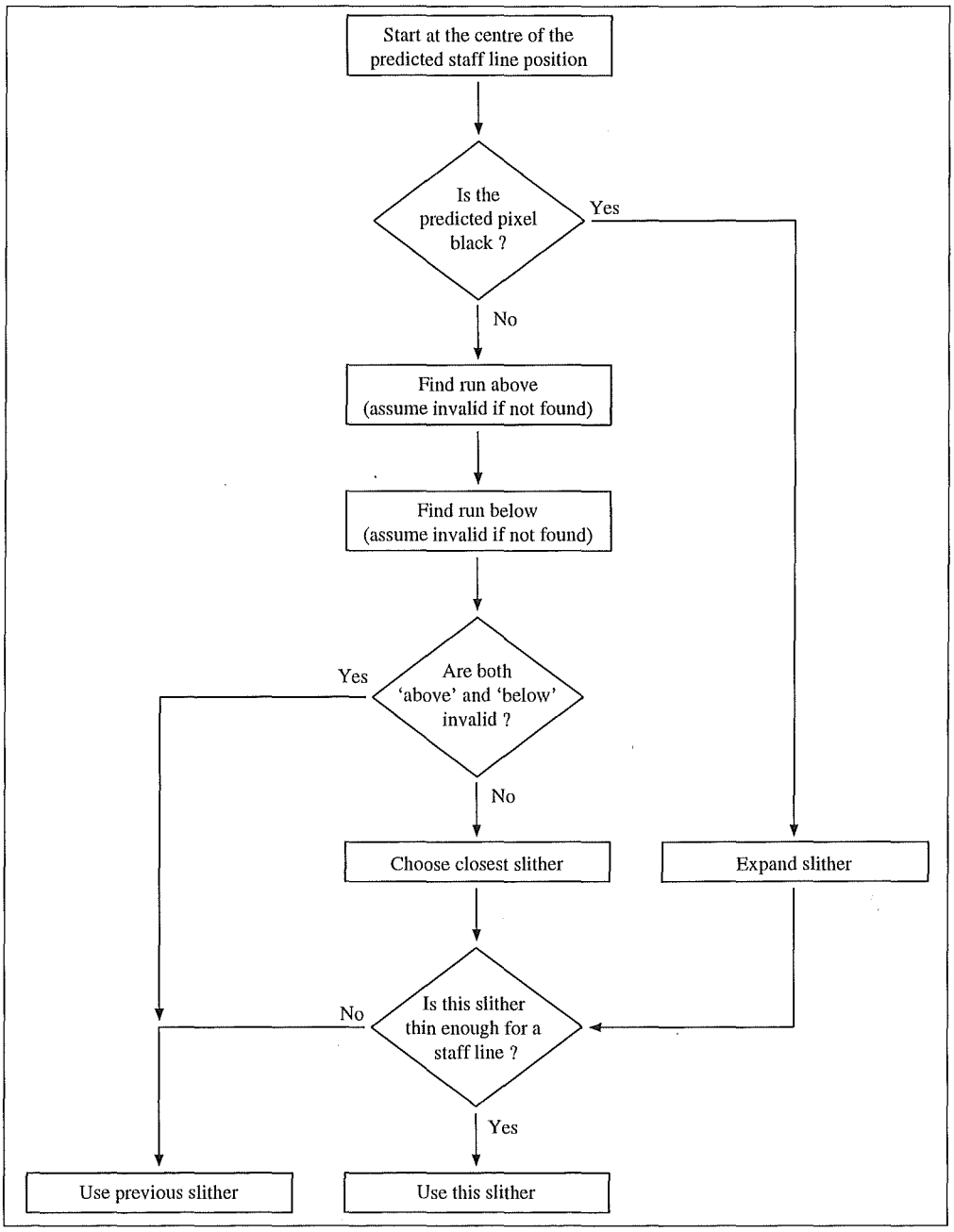


Figure 4.22: The procedure for choosing one slither in the *track* process.

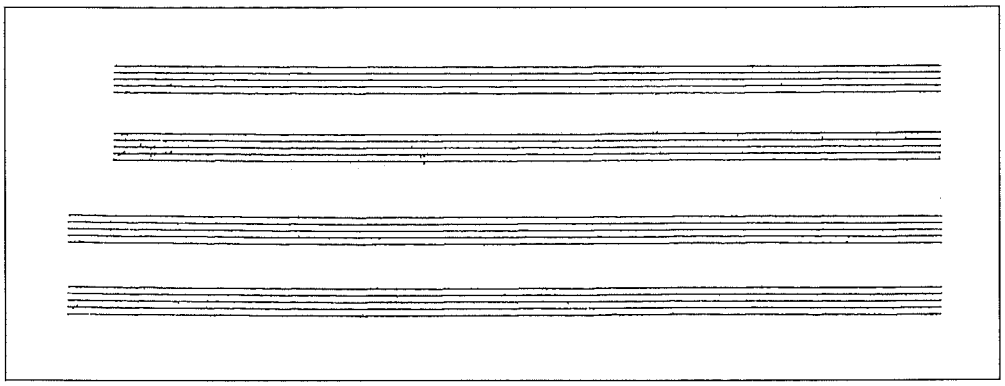


Figure 4.23: The staff lines removed from the example piece of music.

process—the optional image enhancement step of correcting deformation in the y-axis of the image is to be used (Section 5.1), then this information is required. Recording the removed staff lines is not difficult to arrange. The staff line can be built up by inlaying each slither of staff line removed into a bitmap. Figure 4.23 shows the recorded staff lines removed from the example piece (Figure 4.1).

4.3.2 Crossing

In the author's prototype system [Bai91], the algorithm for crossing staff lines relied on an image that had been corrected for skew, hence a one-dimensional boolean array spanning the height of the image was used as a look-up table to determine the lines in the image that corresponded to staff lines. Not only does the above discussion about aliasing effects show the use of the one-dimensional array to be naïve, it also fails to represent staff systems that occur side by side. In addition, we would like the algorithm to be comparable with the staff line removal algorithm, by being capable of processing images that are skew.

All these weaknesses can be overcome by replacing the one-dimensional staff line look-up table, with a two-dimensional array representing the entire image. Thus the algorithm would use `is_staff_line(x,y)` rather than `is_staff_line(y)` to determine the existence of a staff line.

The new look-up array can be generated by assuming that all staff lines are straight, and plotting this information into the array, based on the staff information extracted by the staff detection stage of the OMR process. Alternatively, the imperfect staff lines can be followed (and subsequently plotted in the look-up array) using the same variations de-

veloped for the staff line removal algorithm: *wobble* and *track*.

A further improvement is to generalise the test for an object continuing on the other side of the staff line. The prototype system used the test devised by Clarke *et al.*. However, this is not a rigid requirement and any test devised for a staff line removal algorithm can be utilised.

4.4 Evaluation

When choosing an object location strategy for an OMR system two important considerations are accuracy and speed. In the evaluation section below we first tabulate how well the *wobble* and *track* strategies follow staff lines when compared with a strategy that assumes a staff line is straight. Then we compare the computational cost of these three strategies when used in object location algorithms based on removing, and crossing staff lines.

4.4.1 Accuracy of the staff lines detected

The development of the *wobble* and *track* strategies were motivated by a combination of imperfect staff lines drawn in the original, deformities introduced through scanning, and aliasing effects. In this section we evaluate how well these two strategies follow staff lines.

The accuracy of the staff lines removed was calculated using the "ideal" staff lines generated by hand for the representational cross-section of the corpus in the previous chapter (page 63). Results are shown in Tables 4.2 and 4.3, where accuracy calculations over each image have been vertically subdivided into four equal parts to highlight any increase in error due to an algorithm failing to detect a staff line that bends near the edge of a page.

Even with hand-edited staff lines it is impossible to generate an image that represents the true staff lines, since superimposed objects obscure the upper and lower limits of the line, requiring "educated guesses." Extreme occurrences of this are when beams fall on a section of staff line, or long slurs of low curvature cross a staff line. Consequently, a comparison of the hand-generated "ideal" staff lines with staff lines generated by the computer will always include a residual difference.

A high value in the "missed" column indicates the algorithm does not always follow the staff lines, and increases the number of individual objects that remain connected in the image. A high value in the "extra" column is caused by an algorithm mistaking

Category	Extreme Left			Centre Left		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	63072	19150	3122	80699	16470	1786
Miniature score	4710	1851	785	30192	14585	1124
Accompaniment	121807	45019	3440	167813	112320	637
Monolinear (no chords)	39436	6077	1373	63622	13509	586
Organ	116327	58553	7468	172565	119845	7663
Guitar	64799	39567	1510	58977	48567	139
Piano	45558	13839	1637	58513	11804	520
Popular piano	53575	23078	1208	60208	50596	406
Hymn	43690	11753	1759	52432	17674	686
Percussion	3233	2203	196	6683	6235	33
Computer generated	97680	0	0	103560	0	0
Handwritten	21990	4864	1104	90875	38652	1439
Sol-fa	22949	7614	2629	42689	10905	701
Tablature	64541	27444	2047	71734	46935	216
Square head notation	13359	8978	39	21519	19186	0
Total	776726	269990	28317	1082081	527283	15936

(a)

Category	Extreme Left			Centre Left		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	63072	2333	1950	80699	1394	981
Miniature score	4710	398	862	30192	1499	997
Accompaniment	121807	2050	4193	167813	3386	1244
Monolinear (no chords)	39436	844	1524	63622	1983	594
Organ	116327	3589	3126	172565	3095	1432
Guitar	64799	4516	791	58977	1673	818
Piano	45558	2476	1116	58513	2233	487
Popular piano	53575	716	1411	60208	410	562
Hymn	43690	1213	2020	52432	1280	1417
Percussion	3233	352	202	6683	490	297
Computer generated	97680	0	219	103560	0	77
Handwritten	21990	1244	1005	90875	2999	2684
Sol-fa	22949	782	2453	42689	907	727
Tablature	64541	1321	1867	71734	618	380
Square head notation	13359	536	34	21519	1030	104
Total	776726	22370	22773	1082081	22997	12801

(b)

Category	Extreme Left			Centre Left		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	63072	7295	4736	80699	5909	3411
Miniature score	4710	1326	1729	30192	6572	2595
Accompaniment	121807	3021	7720	167813	6536	3273
Monolinear (no chords)	39436	1774	3916	63622	4763	2726
Organ	116327	32575	11319	172565	24134	10664
Guitar	64799	10211	2140	58977	5047	2917
Piano	45558	4207	4179	58513	4604	3943
Popular piano	53575	2273	3575	60208	2863	2549
Hymn	43690	1933	2694	52432	1921	3066
Percussion	3233	722	642	6683	1225	811
Computer generated	97680	0	5520	103560	0	4713
Handwritten	21990	564	2472	90875	1762	6434
Sol-fa	22949	2844	4325	42689	2650	1477
Tablature	64541	6643	3879	71734	4975	1895
Square head notation	13359	1634	135	21519	2518	441
Total	776726	77022	58981	1082081	75479	50915

(c)

Table 4.2: Accuracy of the staff lines removed from the left half of each image when applied to the representational cross-section of the corpus (a) straight line (b) *wobble* strategy (c) *track* strategy.

Category	Centre Right			Extreme Right		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	80814	13049	1215	71593	26512	1866
Miniature score	30869	21517	585	23162	17793	503
Accompaniment	149307	84170	1780	102926	57832	2345
Monolinear (no chords)	69620	21662	627	56315	22989	544
Organ	174121	88274	4003	155461	65912	2982
Guitar	56840	55263	55	57550	57550	65
Piano	58426	15558	1161	47241	19685	1593
Popular piano	62715	59455	22	52096	52096	0
Hymn	53108	24940	358	46685	26530	185
Percussion	6369	6369	0	7011	7011	0
Computer generated	103440	0	0	95620	0	0
Handwritten	96447	70049	292	77316	63862	587
Sol-fa	47092	14020	667	44481	19132	599
Tablature	70329	58218	70	45808	39420	30
Square head notation	21044	21044	0	4825	4825	0
Total	1080541	553588	10835	888090	481149	11299

(a)

Category	Centre Right			Extreme Right		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	80814	991	852	71593	2547	839
Miniature score	30869	1167	704	23162	592	597
Accompaniment	149307	2142	1787	102926	1449	1372
Monolinear (no chords)	69620	2473	508	56315	2703	655
Organ	174121	2602	1497	155461	2731	1229
Guitar	56840	2083	692	57550	2570	753
Piano	58426	2299	441	47241	2450	745
Popular piano	62715	546	468	52096	377	279
Hymn	53108	893	1281	46685	784	897
Percussion	6369	403	488	7011	825	229
Computer generated	103440	0	210	95620	0	60
Handwritten	96447	3302	1853	77316	3133	1625
Sol-fa	47092	1205	735	44481	2309	595
Tablature	70329	552	428	45808	278	362
Square head notation	21044	616	28	4825	216	8
Total	1080541	21274	11972	888090	22964	10245

(b)

Category	Centre Right			Extreme Right		
	Number of ideal pixels	Missed	Extra	Number of ideal pixels	Missed	Extra
Orchestrated score	80814	3915	3352	71593	7419	3019
Miniature score	30869	6902	2622	23162	4535	1709
Accompaniment	149307	5164	4813	102926	6534	4726
Monolinear (no chords)	69620	7559	3029	56315	6831	2477
Organ	174121	15527	6925	155461	16095	4344
Guitar	56840	4311	3211	57550	5652	1961
Piano	58426	4335	4435	47241	4462	2784
Popular piano	62715	2954	2126	52096	1088	1392
Hymn	53108	1753	2539	46685	1012	1690
Percussion	6369	1486	1020	7011	2077	558
Computer generated	103440	0	5786	95620	0	4636
Handwritten	96447	1579	4097	77316	2064	4055
Sol-fa	47092	3861	1251	44481	5981	1283
Tablature	70329	5494	1932	45808	1878	1368
Square head notation	21044	2346	220	4825	696	17
Total	1080541	67186	47358	888090	66324	36019

(c)

Table 4.3: Accuracy of the staff lines removed from the right half of each image when applied to the representational cross-section of the corpus (a) straight line (b) *wobble* strategy (c) *track* strategy.

other parts of the image as staff line, increasing the number of fragmented objects.

Overall, the *wobble* strategy outperforms the *track* strategy with lower counts for both missed and extra pixels. In terms of extra pixels, following a straight line is comparable with the *wobble* strategy, but the number of pixels missed is far greater than either of the other methods.

4.4.2 Removal

The *wobble* and *track* strategies were added to the template based method described by Clarke *et al.* [CBT88a] and the chord based method described by Martin and Bellissant [MB91]. Application of these variations to the example piece of music are shown in Figures 4.24 to 4.28. The template method following a straight line produces extremely poor results because the staff lines warp up towards the left-hand side of the image. The other four algorithms fare better and produce good results that are similar in quality.

Comparative processor costs are shown in Figure 4.29. The track based methods spend slightly more time than their wobble based counterparts choosing the next slither, and both chord based methods are around 13 times more expensive than their respective template methods for this image. Chord based staff line removal, therefore, is an unattractive solution to musical object location.

4.4.3 Crossing

The same variations constructed for the staff line removal algorithm were tested using the equivalent staff line crossing algorithm. In Figure 4.30 comparative processor costs are shown to the same scale used in Figure 4.29. For this image, the template based crossing algorithms are only marginally faster than their removal based versions, but for the chord based methods we see a dramatic improvement. The chord based crossing methods are 3 times faster than the chord based removal methods. Thus, object location using the more sophisticated check can be reduced to only 5 times more expensive than the template method, as opposed to the previous comparison of 13.

For the example image, the benefits from restricting the expensive check for the existence of a musical object at a given point on a staff line to only those parts of the staff line where an object comes into contact, outweighs the cost incurred in modifying the flood-fill algorithm.

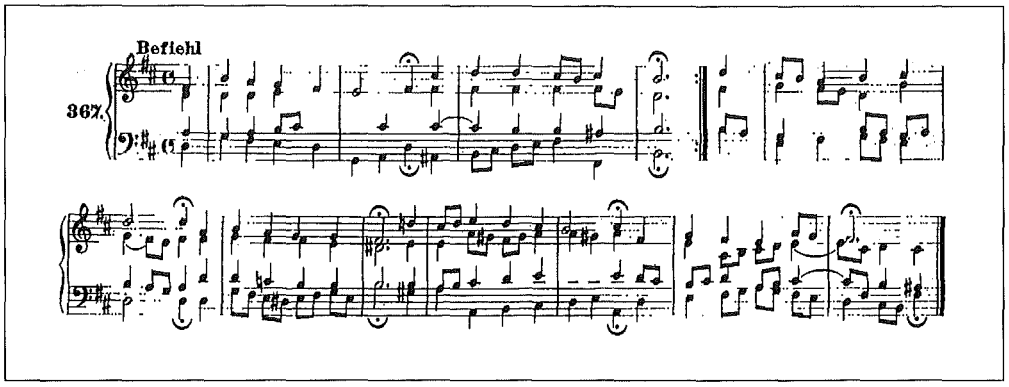


Figure 4.24: Basic staff line removal using a template.

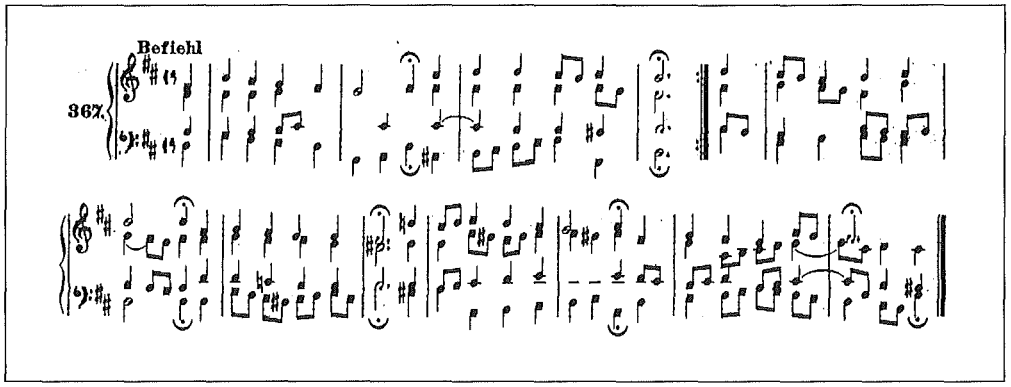


Figure 4.25: *Wobble* added to the template based staff line removal algorithm.

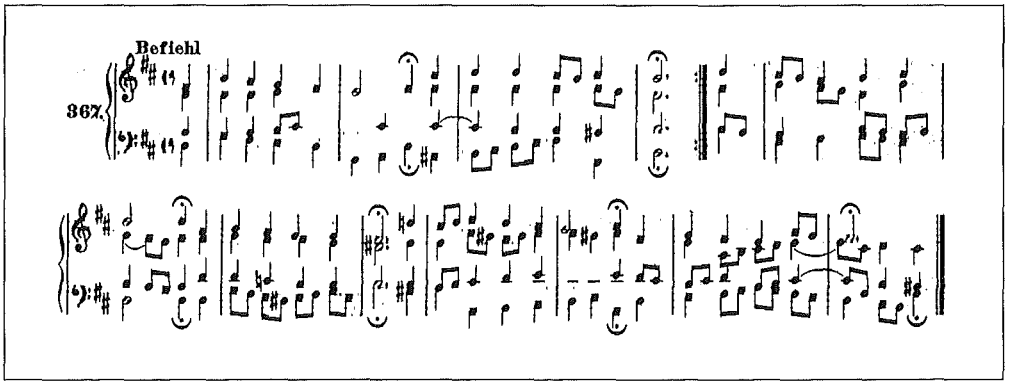


Figure 4.26: *Track* added to the template based staff line removal algorithm.

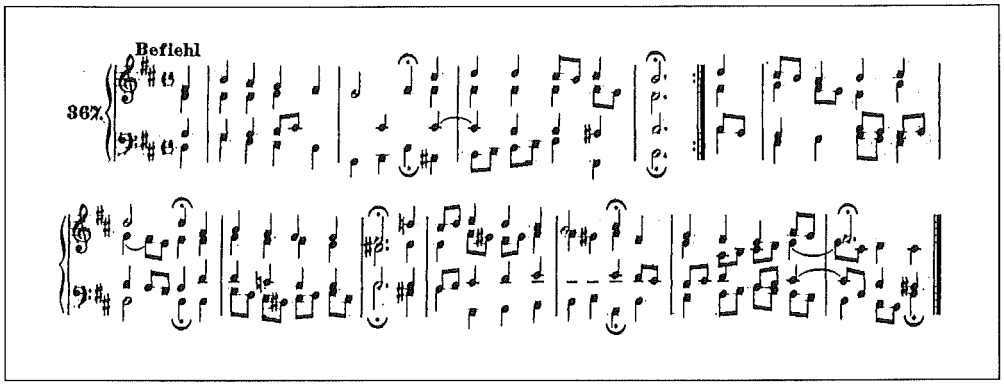


Figure 4.27: *Wobble* added to the chord based staff line removal algorithm.

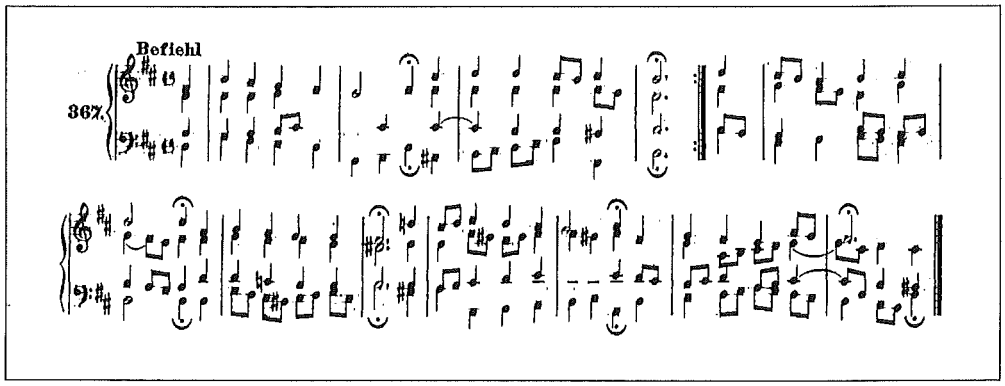


Figure 4.28: *Track* added to the chord based staff line removal algorithm.

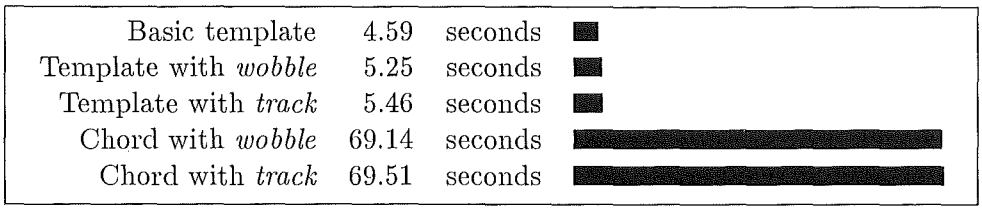


Figure 4.29: Timing information for object location using variations on the staff line removal algorithm when processing the example piece of music.

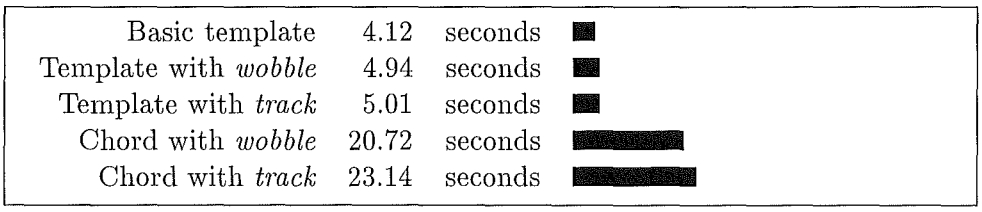


Figure 4.30: Timing information for object location using variations on the staff line crossing algorithm when processing the example piece of music.

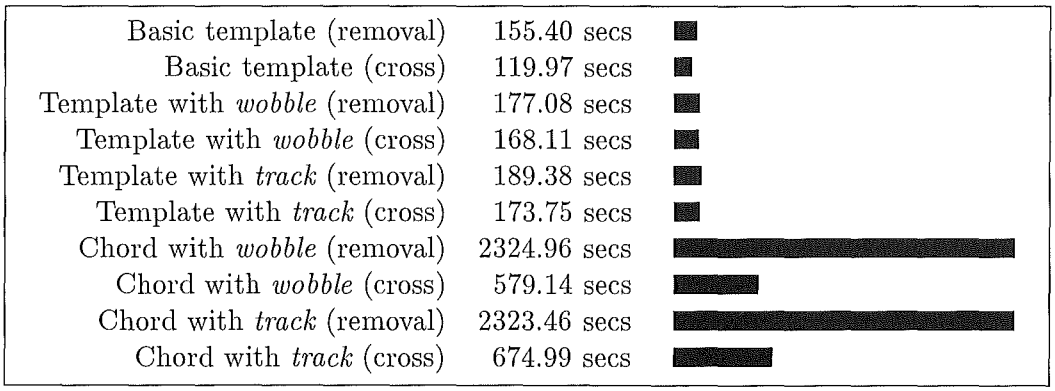


Figure 4.31: Average processing timing for object location when processing the representational cross-section of the corpus.

In a more substantial test, the representational cross-section from the corpus was processed (Figure 4.31). These results confirm the staff line crossing algorithms are faster than the equivalent staff line removal algorithms.

Combining the results on accuracy and speed, we choose the *wobble* version of the template based staff line crossing algorithm for musical object location in subsequent chapters.

Chapter 5

Image enhancement

Image enhancement is a subjective process, where the aim is to improve the quality of an image with respect to its intended use. For example, pictures taken from space of far off galaxies are computer enhanced to reveal details that the human eye cannot naturally see. In OMR work, improvements to the image can be made that simplify subsequent stages in the process. *Correcting the image for skew* means that later stages need not worry about musical artifacts, such as note stems, being off-vertical. *Correcting the image for deformation* means the later stages need not worry about the height of the same note varying in vertical position across the page.

In this chapter, these enhancements are investigated in turn with relevant general image processing techniques being described and discussed. To aid the investigation, variations on basic algorithms were developed to gain quantitative results, and to allow comparisons. In particular, the *speed* versus *vertical error* trade-off between correcting an image for skew by rotation and shearing is analysed; and a method that attempts to correct deformations in a scanned image, based on how the detected staff lines vary away from their expected position, is discussed.

A standard image processing technique is to perform a transform in reverse [Rus92], that is, compute where each pixel comes from, rather than where each original pixel goes to. Designing shearing or rotation algorithms in this way helps reduce the number of white "holes" that can appear due to rounding errors in the forward transform. For simplicity, the algorithms presented in this chapter are described as forward transforms. The algorithms implemented for evaluation, however, use the reverse transform refinement.

5.1 Correction for skew

Correction for skew is possible because earlier processing in the OMR system has identified the angle of the staff lines. Working with a levelled document simplifies later stages in an OMR system, since simpler assumptions can be made, such as all note stems will be vertical.

When OMR research first began it was unclear what, if anything, could be accomplished. Consequently, errors were kept to a minimum, at the expense of increased computation. Correcting the image for skew by rotation is one example of this. Now that more is known about the OMR problem, perhaps a faster approximation for correction is sufficient; perhaps no correction is needed at all. In this section, we assume that correction for skew is still desired, and consequently focus on the options available to us.

5.1.1 Shearing versus rotation

Shearing is a similar operation to rotation (Figure 5.1). Because all the starting coordinates for our application are integral, shearing can be performed using only integer arithmetic:

$$\begin{aligned} \textbf{Shear:} \quad x_s &= x, \\ y_s &= y_o. \end{aligned} \tag{5.1}$$

This contrasts with rotation, which requires floating-point operations to calculate the trigonometric functions:

$$\begin{aligned} \textbf{Rotate:} \quad x_r &= x_o + \delta_x \cos\theta - \delta_y \sin\theta, \\ y_r &= y_o + \delta_x \sin\theta + \delta_y \cos\theta, \\ \text{where} \quad \delta_x &= x - x_o, \\ \delta_y &= y - y_o. \end{aligned} \tag{5.2}$$

Unfortunately though, shearing only corrects skew in one axis. The operation also shrinks the corrected axis and stretches the orthogonal axis, as is shown in Figure 5.2.

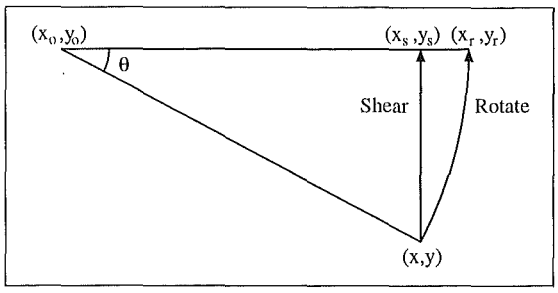


Figure 5.1: Shearing is similar to rotation.

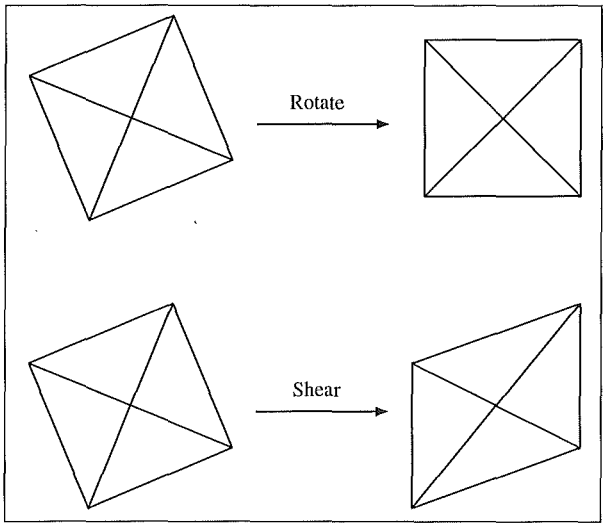


Figure 5.2: Shearing only corrects skew in one axis.

Previous OMR reports have hinted that integer based shearing would be faster than floating-point based rotation, but there have been no published results to corroborate this. Indeed, the issue is more complicated than such cursory comments suggest, since it is increasingly common to find workstations with floating-point arithmetic realised in hardware. Also, optimising the rotation algorithm to use look-up tables and fixed-point arithmetic reduces its computational cost. Ultimately, the choice of correction algorithm depends on hardware limitations and project requirements. The three governing parameters are: *accuracy*, *memory requirements*, and *processor time*. We will now examine these items in turn.

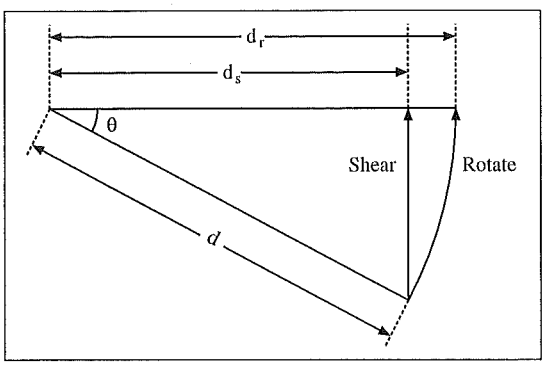


Figure 5.3: The error introduced by shearing.

5.1.2 Accuracy

It was mentioned earlier that the shearing operation shrinks the axis that is being corrected for skew, and stretches the orthogonal axis. Here we analyse this scaling mathematically.

Consider Figure 5.3, which shows how a line of length d is shortened by a shearing operation. The scale factor for the principal axis (in this case the x-axis) is:

$$\begin{aligned} d_r &= d, \\ d_s &= d.\cos\theta, \\ \text{Principal Scale factor} &= \frac{d_s}{d_r} = \frac{d.\cos\theta}{d} = \cos\theta. \end{aligned} \tag{5.3}$$

By similar reasoning, the scale factor for the perpendicular axis (in this case the y-axis) is:

$$\text{Perpendicular Scale factor} = \frac{1}{\cos\theta}. \tag{5.4}$$

Using these two equations, we can correct the errors introduced by a shearing operation. However, a single shearing operation applied in one axis does not correct the skew in the orthogonal axis. What is required is a shear, scale, shear, scale combination, where the two shearing operations are in orthogonal axes. The values required to perform this transformation are not as straightforward as they might first seem. The formula that must be solved is:

$$\begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 \\ 0 & \frac{1}{\cos\theta} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \begin{bmatrix} Sc_x & 0 \\ 0 & Sc_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.5}$$

where:

Sh_x and Sh_y are the shear values in the x and y axes respectively, and

Sc_x and Sc_y are the scale factors in the x and y axes respectively.

This yields the values:

$$\begin{aligned} Sh_x &= -\tan\theta, \\ Sh_y &= \frac{\sin\theta.\cos\theta}{1-\sin\theta}, \\ Sc_x &= 1-\sin\theta, \\ Sc_y &= \cos^2\theta. \end{aligned} \tag{5.6}$$

Performing this shear, scale, shear, scale guarantees that the image is corrected for skew in all directions; however, the computational efficiency that we gained by replacing the floating-point based rotation operation with two integer based shearing operations is lost because the two scaling operations require floating-point arithmetic.

The overhead due to scaling is reduced by rearranging the shear and scale operations. An equivalent transformation is to perform two shearing operations in orthogonal directions followed by one scaling operation. In this case, the formula that must be solved is:

$$\begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \begin{bmatrix} Sc_x & 0 \\ 0 & Sc_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.7}$$

where:

Sh_x and Sh_y are the shear values in the x and y axes respectively, and

Sc_x and Sc_y are the scale factors in the x and y axes respectively.

This yields the values:

$$\begin{aligned} Sh_x &= -\tan\theta, \\ Sh_y &= \cos\theta.\sin\theta, \\ Sc_x &= \frac{1}{\cos\theta}, \\ Sc_y &= \cos\theta. \end{aligned} \tag{5.8}$$

A clever scheme by Paeth [Pae86] decomposes the rotation matrix into three shearing operations, and thus requires only integers. The equation to be solved is:

$$\begin{bmatrix} 1 & Sh_{x1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \begin{bmatrix} 1 & Sh_{x2} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.9}$$

where:

Sh_{x1}, Sh_{x2} and Sh_y are the shear values in the x and y axes.

The corresponding values to this equation are

$$\begin{aligned} Sh_{x1} &= -tan(\theta/2), \\ Sh_y &= \sin\theta, \\ Sh_{x2} &= -tan(\theta/2). \end{aligned} \tag{5.10}$$

The resultant image is scale invariant.

Given that a human operator can scan music "reasonably" level, the amount of crookedness introduced is small. In the corpus of music scanned for this thesis, staff line skew ranges from $-1.05°$ to $+1.19°$, with a standard deviation of 0.32. The errors, therefore, that result in *not* scaling a sheared image are minimal, and an OMR system may prove tolerant of them.

5.1.3 Memory requirements

The process of correcting an image for skew takes a source bitmap and produces a destination bitmap. A straightforward implementation of either the rotation operation or the

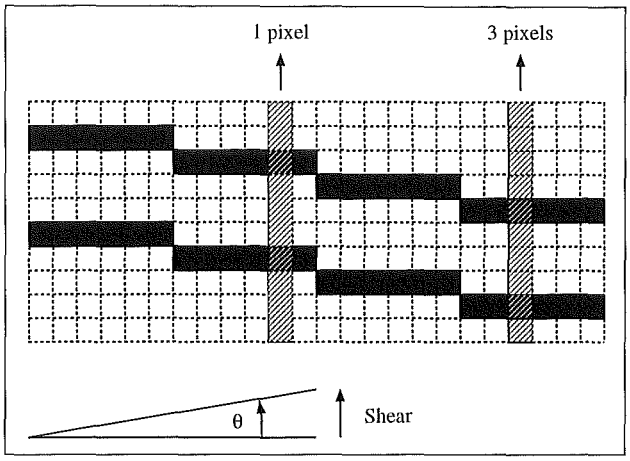


Figure 5.4: In a shearing operation entire columns of pixels move up or down.

shearing operation, therefore, requires a couple of integer variables for loop counters, a few working variables for the inner loop calculations (floating-point in the case of rotation, and integers in the case of shearing), and two bitmaps. An A4 page is 8.25×11.67 inches. Scanned at 300 dpi and storing 8 pixels per byte, a computer needs 1,085,056 bytes to store the image, or roughly one megabyte. Integer and floating-point variables require only a few bytes. Clearly the bitmaps dominate the memory usage.

Fortunately, both rotation and shearing can be implemented using the same bitmap for the source and destination, effectively eliminating any extra memory requirements for this stage of an OMR system.

Shearing with one bitmap

Consider Figure 5.4, which shows a shearing operation that corrects the x-axis.[1] All the pixels in a column are moved the same distance either up or down. By carefully choosing the starting end of the column, based on the direction of the shift, the column can safely overwrite itself.

- To move a column *up*, start at the *top*.
- To move a column *down*, start at the *bottom*.

[1]The same discussion holds true for a shearing operation that corrects the y-axis.

If the bitmap remains the same size, the shearing operation will cause a small amount of the image to "fall off" the edge of the bitmap. The area lost is given by:

$$\text{Area lost by shearing} = \frac{1}{2} x\_dim^2 \tan\theta \text{ pixels}, \tag{5.11}$$

where:

x_dim is the x-dimension of the bitmap in pixels, and

θ is the angle of skew.

An identical number of white pixels are introduced.

The loss is small. An A4 page that is 0.5° skew loses only 0.31% of the bitmap, most of which will already be white. Additionally, by choosing the centre of the shearing operation to be the middle of the page, the lost pixels are evenly distributed around the four corners.

Rotation with one bitmap

A similar memory optimisation is possible for rotation. Consider Figure 5.5, which shows a rotation operation from the source bitmap to the destination bitmap. Each line in the source bitmap is translated though x and y to reach the destination bitmap—some example lines are shown in the figure. By carefully choosing both the direction each line is processed *and* the sequence in which the lines are moved, only one bitmap is needed.

- For a *clockwise* rotation, start at the *bottom* of the bitmap and process each line *from left to right.*
- For an *anti-clockwise* rotation, start at the *top* of the bitmap and process each line *from right to left.*

Similar to the shearing operation, the rotation operation will cause small amounts of the image to "fall off" the edge of the bitmap if it is not resized. The area lost is given by:

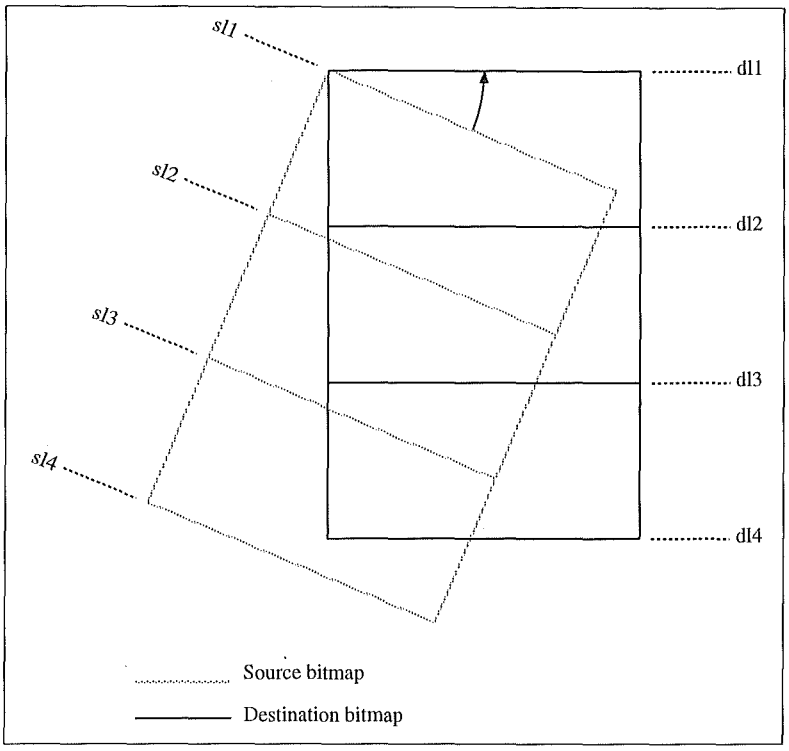


Figure 5.5: Rotating the source bitmap to produce the destination bitmap.

$$area_{x\_tri} + area_{y\_tri} - area_{overlap} \text{ pixels}, \tag{5.12}$$

where:

$$\begin{aligned}
area_{x_tri} &= \tfrac{1}{2}.x_dim^2.\sin\theta, \\
area_{y_tri} &= \tfrac{1}{2}.D^2.\tfrac{1}{\cos\theta.\sin\theta}, \\
area_{overlap} &= \tfrac{1}{2}.x_dim^2.\tfrac{(1-\cos\theta)^2}{\cos\theta.\sin\theta}, \\
D &= y_dim.\sin\theta - x_dim(1-\cos\theta), \\
x_dim &\text{ is the } x\text{-dimension of the bitmap in pixels,} \\
y_dim &\text{ is the } y\text{-dimension of the bitmap in pixels, and} \\
\theta &\text{ is the angle of skew.}
\end{aligned}$$

An identical number of white pixels are introduced.

Once again the loss is small, though larger than the loss from the shearing operation. An A4 page that is 0.5° skew looses only 0.92% of the bitmap. As with shearing, choosing the centre of rotation to be the middle of the page means that the lost pixels are evenly distributed around the four corners.

5.1.4 Processor time

In this section we evaluate the computational cost of various configurations of the shearing and rotation algorithms. We have already seen that a rotation operation can be coded using three shearing operations, requiring only integer arithmetic. Since the angles of skew in OMR applications are typically small, another potential optimisation in the shearing algorithm is to move bytes rather than bits. This modification, however, cannot be made to the rotation algorithm.

Due to the nature of music notation, a high proportion of a piece of music is white space. A standard algorithm corrects an entire page, and consequently a lot of processor time is spent moving "nothing." In the OMR system described in this thesis, the angle of skew is determined after all isolated objects on the page have been categorised as either staff system or not (Section 3.2.1), and so the same effect can be achieved by correcting each of these objects. Alternatively, by incorporating the gradient into the staff line removal algorithm (Section 4.3) a staff system can be decomposed into its staff lines and staff objects which are then corrected, further pruning away white space.

In this section we compare *rotation*, *shearing*, and *1-byte shearing* algorithms. Each of these algorithms are used to correct *the entire page*, *objects after staff detection*, and *objects after staff line removal.*

Evaluation

"Wake Up, Little Susie," shown in Figure 5.6, was corrected completely for skew using rotation (Figure 5.7) and corrected for skew in the x-axis through shearing (Figure 5.8). Their respective horizontal projections are shown in Figures 5.9 to 5.11. Both algorithms work well, transforming an image where peaks due to staff lines in its horizontal projection are indistinct, into one where each peak due to a staff line can be clearly seen.

Figure 5.12 shows the timing results of the various algorithms when applied to the example image. As we anticipated, shearing is faster than rotation, and processing the objects within the page is faster then processing the entire page. A better profile is given in Figure 5.13 which shows timing information for processing the representational cross-section of the corpus.

Table 5.1 gives the timing information for the individual pieces in the cross-section.

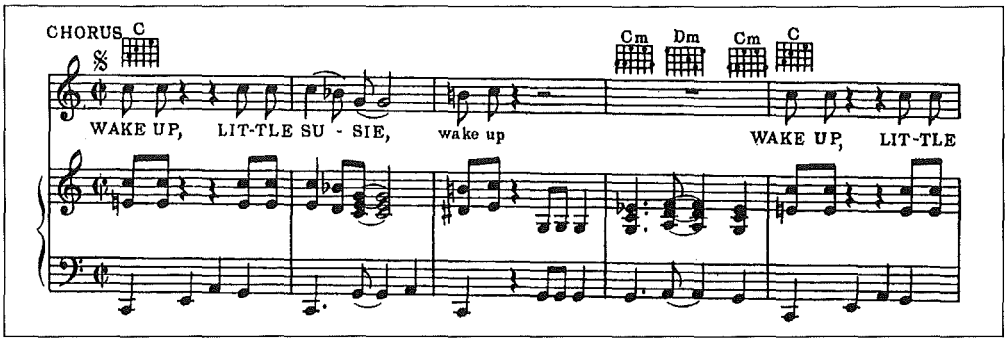


Figure 5.6: An example excerpt of music scanned skew. The music slopes down from left to right.

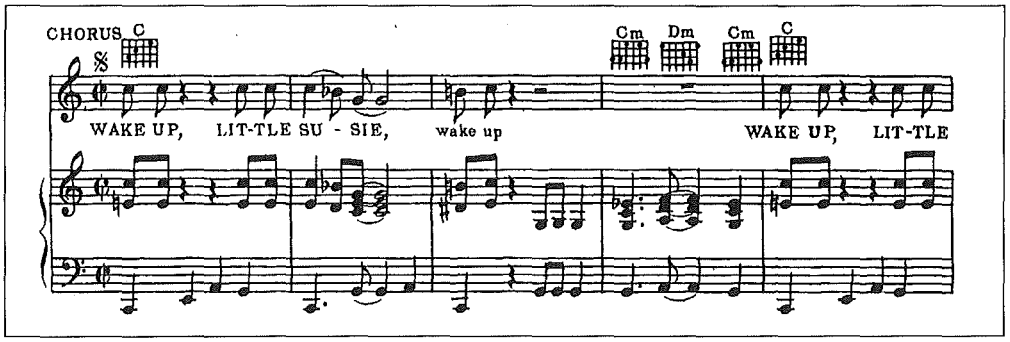


Figure 5.7: The example excerpt of music corrected completely for skew using rotation.

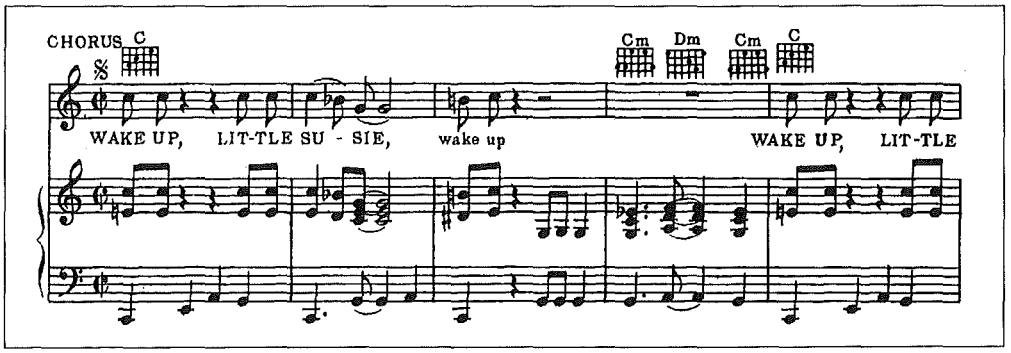


Figure 5.8: The example excerpt of music corrected for skew in the x-axis through shearing.

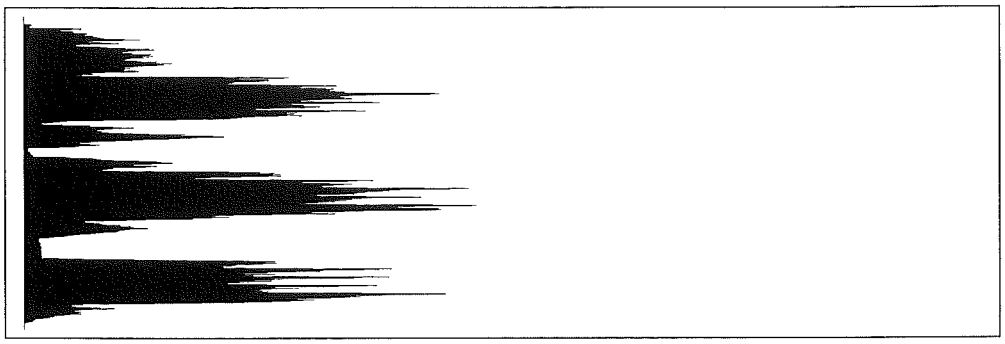


Figure 5.9: The horizontal projection for the example excerpt of music.

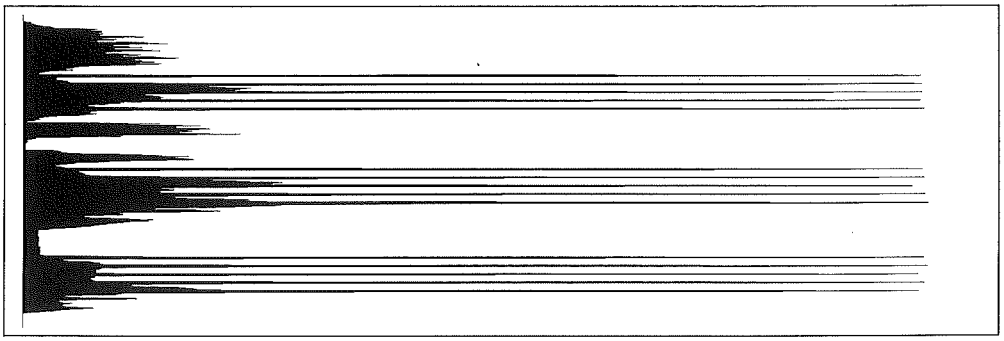


Figure 5.10: The horizontal projection for the example excerpt of music corrected completely for skew using rotation.

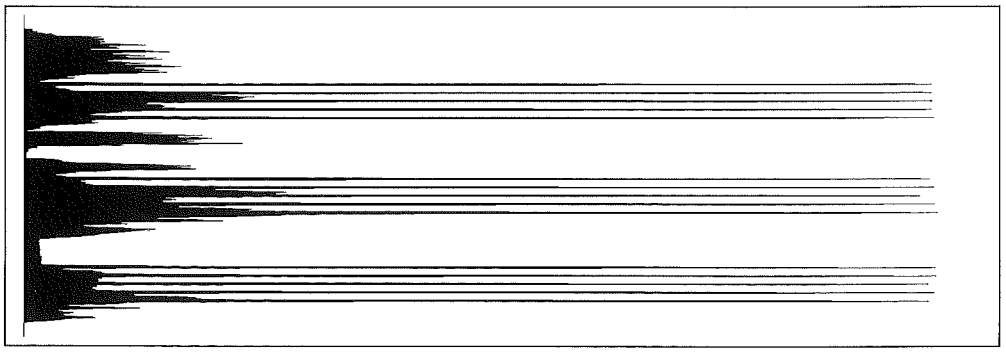


Figure 5.11: The horizontal projection for the example excerpt of music corrected for skew in the x-axis through shearing.

Category	Area processed (pixels)	Rotate (msecs)	Shear (msecs)	Shear bytes (msecs)
Orchestrated score	7750560	28970	19070	2110
Miniature score	3352702	12350	8250	920
Accompaniment	8945400	32890	21830	2730
Monolinear (no chords)	4044504	14500	9950	1110
Organ	8260144	29520	20290	2500
Guitar	7751696	27720	19030	2240
Piano	8415000	31260	22400	2520
Popular piano	7881292	29100	19390	2250
Hymn	2396416	8650	5890	670
Percussion	1232491	4520	3040	380
Computer generated	8797500	32650	21840	2230
Handwritten	3940352	14260	9690	1280
Sol-fa	3781440	14180	9270	2040
Tablature	6484790	24020	15930	1770
Square head notation	3721942	13410	9140	1250
Total	86756229	318000	215010	26000

(a)

Category	Area processed (pixels)	Rotate (msecs)	Shear (msecs)	Shear bytes (msecs)
Orchestrated score	5253041	19260	13180	1620
Miniature score	1663493	6150	4280	660
Accompaniment	5697660	21200	14410	2150
Monolinear (no chords)	2434552	9180	6130	800
Organ	6418733	24150	15980	2150
Guitar	5464175	20520	13670	1780
Piano	5458233	20360	13650	1780
Popular piano	4715610	17420	11950	1740
Hymn	1774551	6630	4520	540
Percussion	879179	3300	2230	300
Computer generated	6801468	24860	17140	1940
Handwritten	2122577	8300	5520	830
Sol-fa	2721032	13440	7100	1640
Tablature	4707541	17120	11870	1470
Square head notation	1765841	7120	4640	800
Total	57877686	219010	146270	20200

(b)

Category	Area processed (pixels)	Rotate (msecs)	Shear (msecs)	Shear bytes (msecs)
Orchestrated score	2800640	15170	10070	1910
Miniature score	469120	3460	2330	680
Accompaniment	1346807	19230	11970	2190
Monolinear (no chords)	913966	7150	4480	950
Organ	1391122	25350	15590	2440
Guitar	2510119	14300	9120	1610
Piano	3327631	15900	10420	1670
Popular piano	1544322	12030	7750	1550
Hymn	576409	5100	3310	660
Percussion	201908	1770	1080	280
Computer generated	2554941	12830	9000	1720
Handwritten	973972	9030	5550	1050
Sol-fa	777316	17890	8280	2150
Tablature	1317893	9830	6330	1270
Square head notation	1065969	5940	3740	780
Total	21772135	174980	109020	20910

(c)

Table 5.1: Time taken to correct skew in the representational cross-section of the corpus (a) correcting the entire page (b) correcting the objects after staff detection (c) correcting the objects after staff line removal.

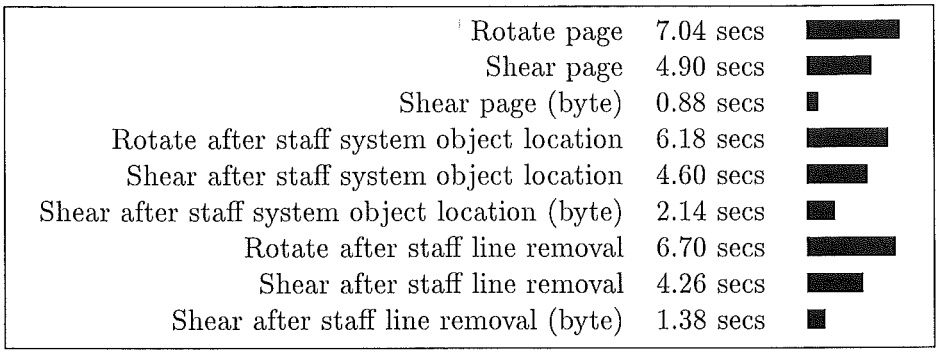


Figure 5.12: Timing information for correcting skew in the example image.

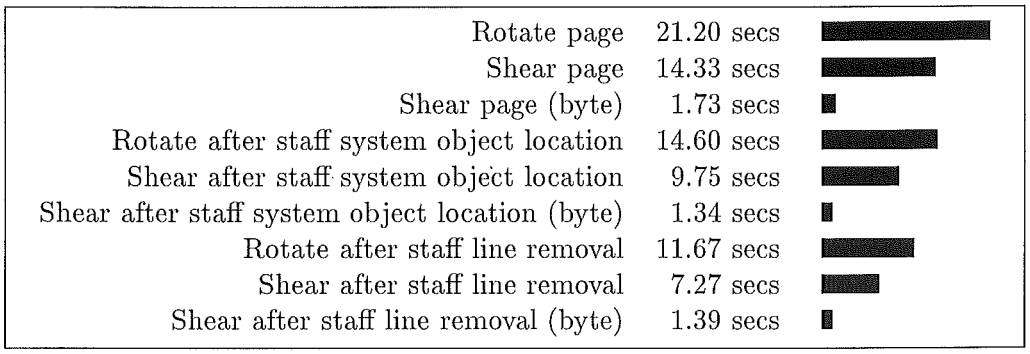


Figure 5.13: Average time taken to correct skew in the representational cross-section of the corpus.

As we would expect, the area requiring correction for skew reduces as the OMR system cuts away more of the white space. Correcting for skew after staff system location reduces the area processed by 33%, and delaying correction until after the staff lines have been removed results in an overall reduction of 75%. Nevertheless, such reductions do not necessarily imply improved performance. For example, when processing the first five entries in the representational cross-section, shearing with byte operations after staff line removal, (Table 5.1c) is slower than applying the same algorithm after staff system location (Table 5.1b). This is due to an overhead in the algorithm that requires the first and last bytes in each line to be treated separately. As the width of the objects to be processed become smaller—as is the case with removing the staff lines—this overhead starts to dominate the computational cost of the algorithm.

An application of a shearing algorithm only corrects skew in one axis. To completely correct skew in an image, three shearing operations are required (Section 5.1.2, page 102).

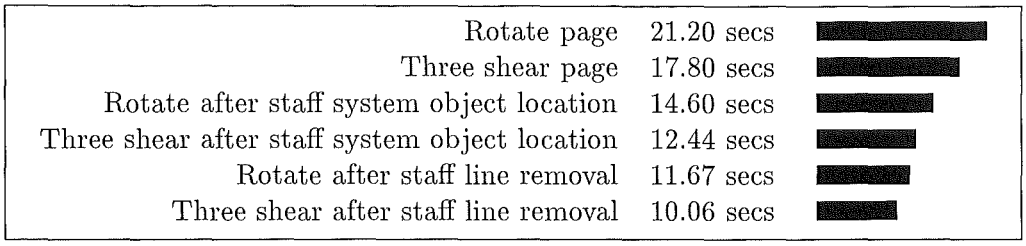


Figure 5.14: Based on processing the representational cross-section of the corpus, extrapolated timing information for completely correcting an image for skew.

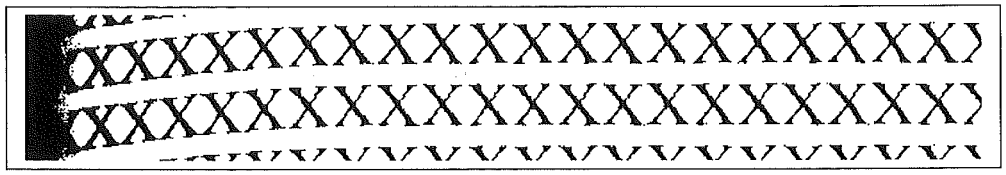
Extrapolating the results from Figure 5.13, Figure 5.14 shows how long the various algorithms would take to completely correct an image for skew. For the shearing algorithm, the byte optimisation is only possible when correcting skew in the x-axis, and therefore the best equivalent shearing combination that is equivalent to a rotation is: byte-based, bit-based, byte-based.

Overall, each successive optimisation lowers the computational cost of the transform, with three shearing operations after staff line removal 53% faster than standard algorithm that rotates the entire page. But as we saw above, for individual images this speed up is not always the best. When transforming the sample piece of handwritten music, for example, shearing the objects after staff system location is faster (7180 msecs) than processing after staff line removal (7650 msecs).

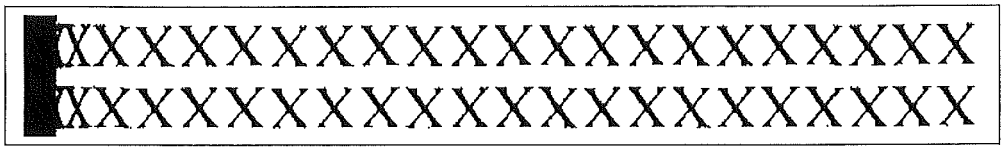
5.2 Correction of deformation

In Section 1.4 we reviewed existing scanner technology, and chose the flatbed style of scanner for this project. Here we concentrate on the types of deformation caused by the physical design of this class of scanner. This does not, however, restrict the application of solutions to music acquired using a flatbed scanner, as other scanner types can cause the same types of deformation.

Deformation in a scanned image occurs when the original document does not lie flush on the scanning surface. The same is true for photocopies. Such deformations are visually noticeable in scanned works that come from a tightly bound book, with the familiar curving lines of text resulting from a book that could not be pressed flat enough against the photocopier.



(a)



(b)

Figure 5.15: Deformation caused by scanning from a tightly bound book depends on the orientation of the spine (a) the spine is parallel to the scan line (b) the spine is perpendicular to the scan line.

The form of deformation depends on the orientation of the spine with respect to the scanning device. To illustrate this, a uniform grid of the letter X was printed on a page which was then slotted into a tightly bound book and scanned both horizontally and vertically. The results of this are shown in Figure 5.15. In Figure 5.15a the "classic" bowing form of deformation is shown, where each row curves down at the left-hand side as the page nears the spine, and consequently is raised away from the scanning surface. In Figure 5.15b a compressing effect is observable, where the width of each letter X narrows as it approaches the spine on the left-hand side. The orientation of the spine on the scanning surface is normally dictated by the dimensions of the book.

A scanned piece of music with compressed staffs needs correction in the x-axis. This form of deformation, however, is unlikely to produce errors in an OMR system. A staff line ending prematurely does not affect the ability of an OMR system to locate staff lines, or remove them, and increased emphasis in recognition routines can be given to vertical properties of musical objects such as position and dimension, which are unaffected by the compression. Musical features most at risk are clefs, key signatures and time signatures. In this thesis, no attempt to correct this form of deformation was made.

A piece of music that has bowing staff lines needs correction in the y-axis, otherwise musical features will be found out of position. For instance, in a staff with lines bowing down, a semibreve rest may be incorrectly classified as a minim rest, and the pitch of

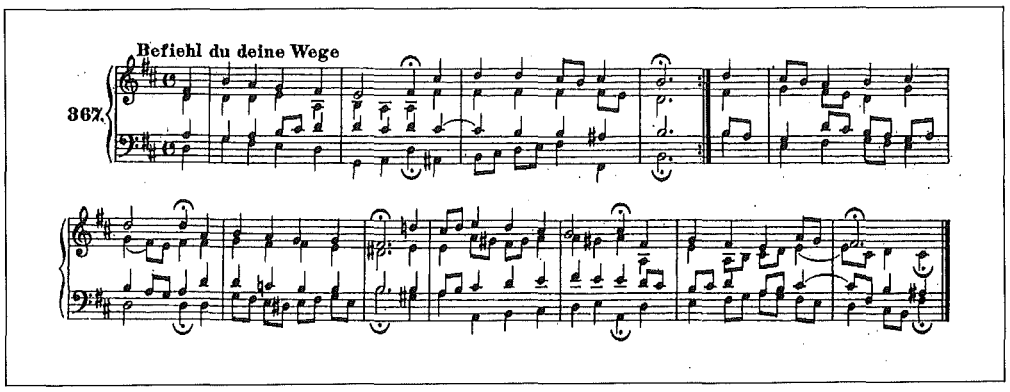


Figure 5.16: The example piece of music previously used to illustrate musical object location techniques. The piece warps upwards at the left-hand side.

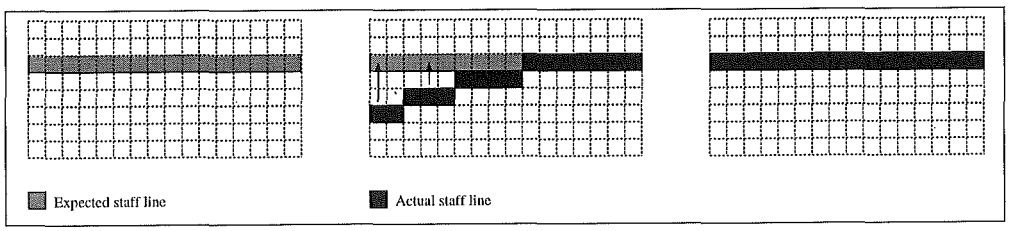


Figure 5.17: How a theoretically horizontal staff line appearing in a scanned image represents the vertical deformation in that part of the page.

a note may be incorrectly calculated as one tone lower than it should be.

5.2.1 Correcting deformation in the y-axis

To show the improved tolerance of the refined musical object location algorithms in Chapter 4, an example was deliberately used where the staff lines bowed up at the left-hand side. This example is reused here (Figure 5.16) to illustrate a methodology that corrects deformation in the y-axis.

The basic algorithm

Staff lines in a piece of music are supposed to be horizontal. Any deviation in the vertical position of the actual staff lines present in a scanned image, therefore, explicitly defines the deformation for that part of the image. This point is illustrated in Figure 5.17.

In a typical piece of music, staff lines collectively cover a large proportion of the

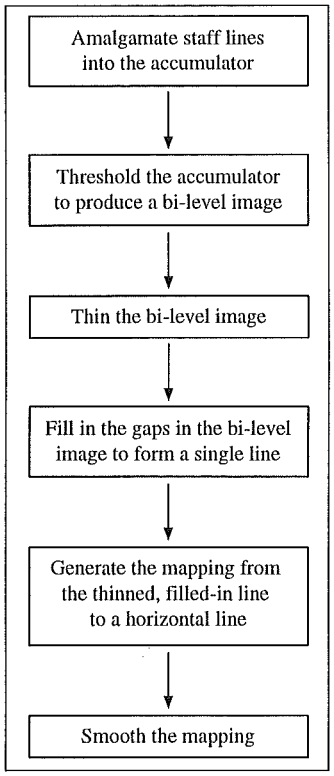


Figure 5.18: An overview of the steps involved with correcting an image for deformation in the y-axis.

width of the image. Accordingly, staff lines offer a convenient way of detecting the overall vertical deformation present in an entire image. Because the deformation recorded by a staff line is local to that part of the page, an algorithm that corrects an entire page for vertical deformation must somehow diffuse this information throughout the image. Figure 5.18 shows an overview of a basic algorithm based on this observation. Figure 5.19 illustrates each step.

First, the located staff lines are collapsed in the y-direction into an accumulator, which records how many black pixels from each staff line coincide. This amalgamated staff line is then subjected to thresholding to reduce the information into a bi-level bitmap. The bitmap is then thinned into line segments and any gaps filled in. Finally the accumulated processed bitmap line is translated into a discrete mapping in the y-axis, which is then smoothed to reduce local unevenness.

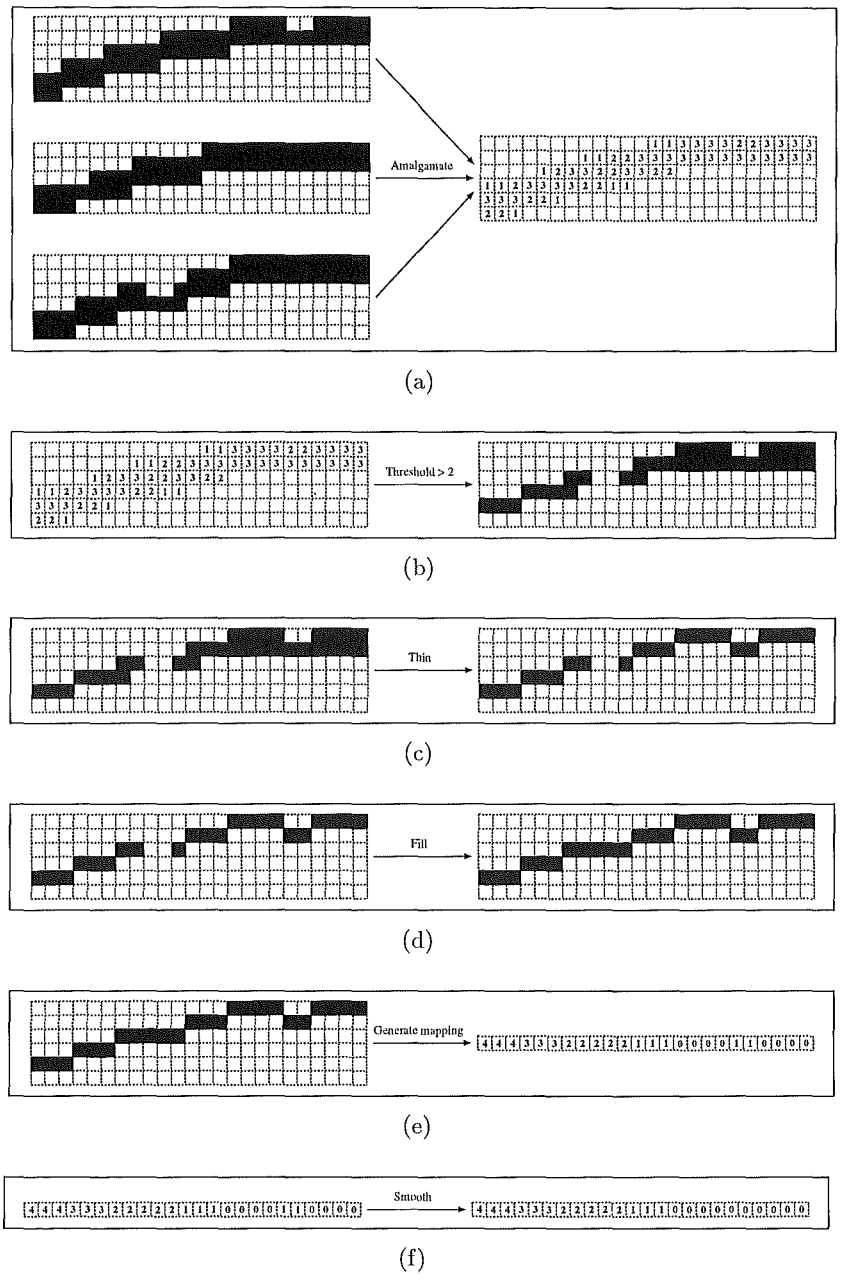


Figure 5.19: A pictorial account of the steps carried out by the y-axis undeformation algorithm.

These steps are elaborated below.

Thresholding. The threshold level is determined as a proportion of the maximum value appearing in the accumulator. For experimentation, various values are available: 60%, 70%, and 80% of the maximum.

Thinning. A simple solution is used: for each column in the bi-level bitmap the highest black pixel and lowest black pixel are found, and the mid-point between these two pixels used as part of the shape's skeletal backbone.

Filling in gaps. It is possible for discontinuities in both the x and y directions to occur in the thinned line. Jumps in the y direction are allowed, but any gaps in the x direction are filled in by plotting a straight line between the two segments.

Smoothing. Local unevenness in the transform is smoothed out by replacing all entries in the array by the average of its own value, and its two neighbouring entries. The first and last entries in the array remain unchanged.

Once the transform has been generated, the image can be corrected by displacing each pixel in the image by the amount shown in the corresponding x position of the mapping.

Non-homogeneous bending

So far we have described the basic algorithm for correcting the deformation in the y-axis, but there is one last twist to the problem. The amount of bending caused by the scanning process gradually changes along the length of the scan. Figure 5.20 shows an example scan from a book where the length of the spine is marginally shorter than the width of the scanner window. In Figure 5.21 enlarged excerpts from the same scan are shown, one from the left-hand side and the other from the right-hand side. The bending seen in these two excerpts are opposite in curvature.

The final stage of the algorithm, therefore, is to generate two discrete mappings using the basic algorithm: one for the top one-third of the image, and the other for the bottom one-third of the image. Pixels are corrected by indexing the two mappings based on the pixel's x co-ordinate, and interpolating between these two values based on the pixel's y co-ordinate to determine the distance the pixel needs to be moved.

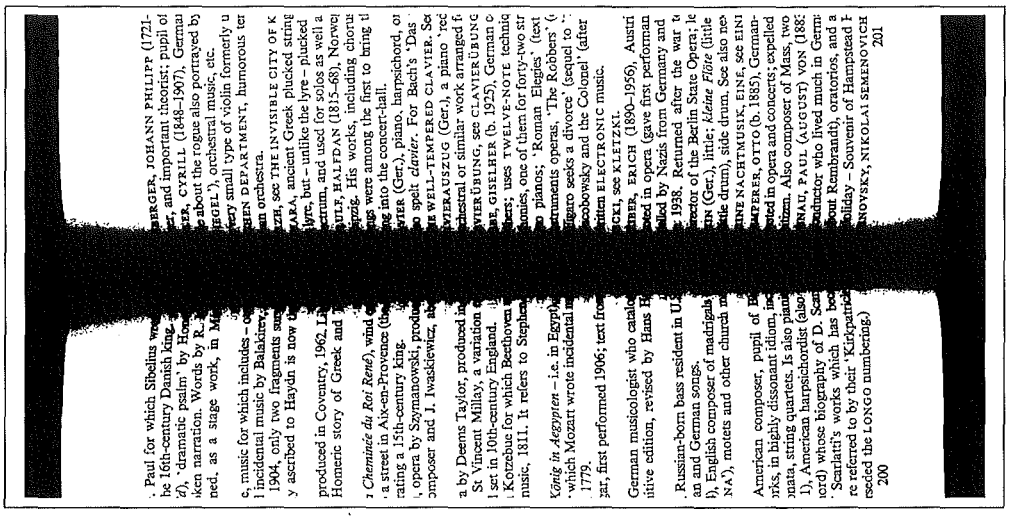
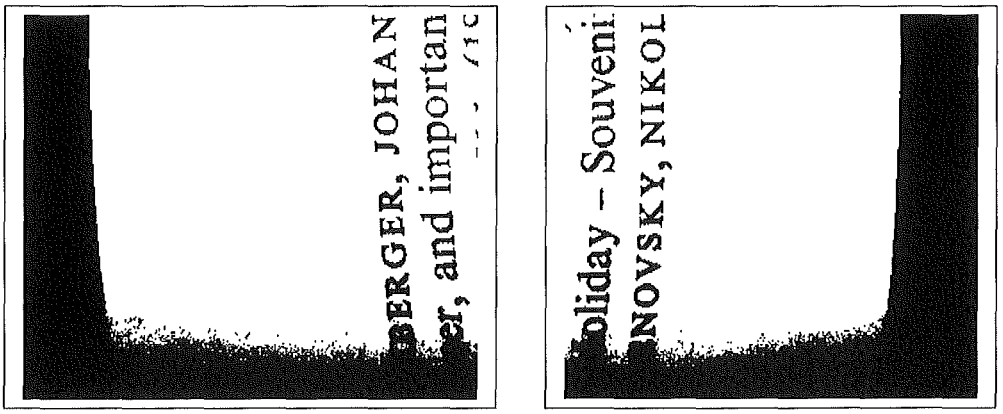


Figure 5.20: The amount of bending gradually changes along the length of the scan.



(a) (b)

Figure 5.21: Enlarged excerpts from the same scan showing the changes in bending (a) the left-hand side (b) the right-hand side.

The solution is robust. A scanned image is not necessarily taken from the entire scanner window—for example, in the case of a scanner operator cropping the top half of the window, the resultant image would be strongly curved at one end and flat at the other. Even so, the algorithm will still obtain satisfactory results, since the interpolation is still true to the gradual change in deformation.

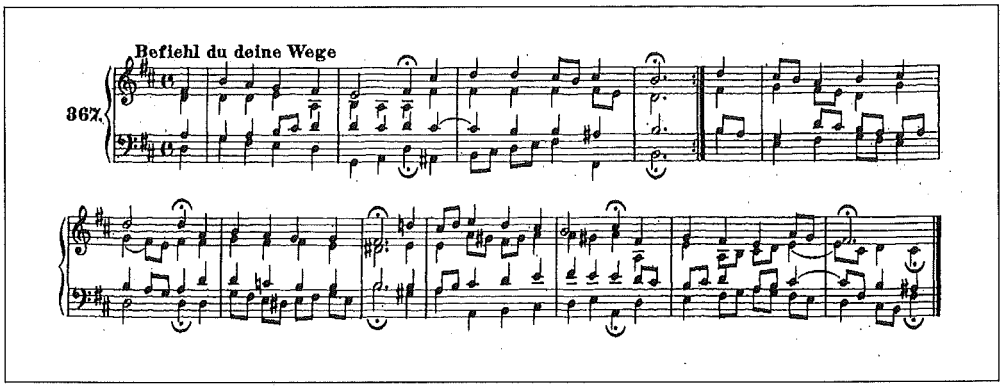


Figure 5.22: The example piece of music after it has been corrected for deformation in the y-axis using an accumulator threshold of 80%.

5.2.2 Evaluation

The result of applying the algorithm for the correction of deformation in the y-axis to the example image (Figure 5.16) is shown in Figure 5.22. Visually there seems little change. The improvement in the image is better demonstrated by comparing the horizontal projections of the deformed and corrected images. Their respective projections are shown in Figure 5.23a and Figure 5.23b, where the latter has sharper, higher peaks. This improvement can been seen in the average height, in the two projections, of lines above 50% of the width of the image. In the uncorrected image, the average is 1396 pixels; in the corrected image this value has been increased to 1587 pixels.

5.3 Observations

The algorithm presented for correcting deformation is a generalised form of shearing. If an image were scanned skew and distorted, then the algorithm to correct deformation would not only unwarp the original piece, but it would also automatically correct any skew in the horizontal axis. With this work it may prove possible to circumvent the correction for skew stage in the OMR process. The key issue is whether or not an OMR system can successfully complete the remaining stages in the process, with skew remaining in the vertical axis.

Results presented in Chapter 9 indicate this to be true. In this chapter a comparison of two OMR configurations is given. First we process 11 images using an OMR system based on rotation that corrects skew in both axes. Then we repeat the experiment

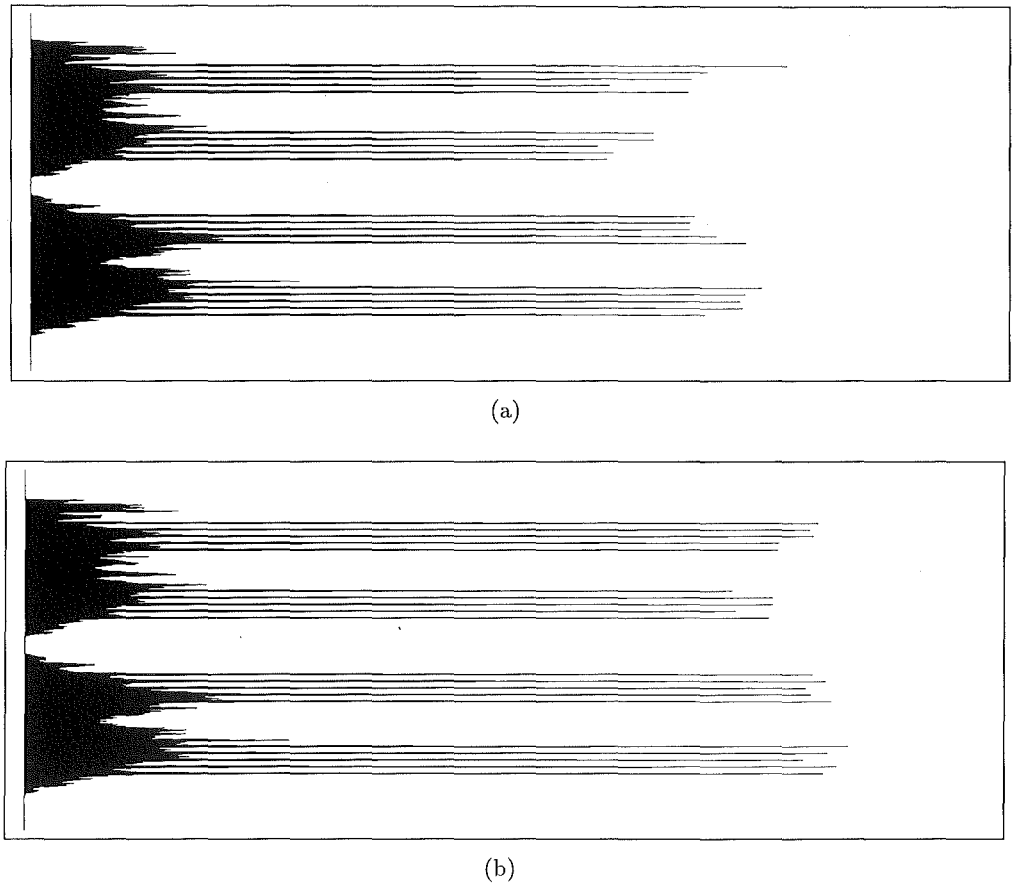


Figure 5.23: A comparison of horizontal projections (a) before the y-axis deformation correction algorithm was applied (b) after the algorithm was applied.

using an OMR system based on a single shearing operation that only corrects skew in the horizontal axis. The comparison reveals near identical results.

An additional form of image enhancement not investigated in this thesis is the elimination of noise from the scanned image. Numerous filtering methods exist [Cas96, MW93], but none were implemented. There were two reasons for this. First, due to the wide scope of the project, including images with excessive noise would only reduce the clarity of results when testing other sections of the OMR system. Second, when a thin part of a shape in an image like a note stem starts to break up—due to either poor quality in the original or a badly set scanning intensity—it is possible for noise filtering techniques to accidentally remove the fractured parts, mistaking them for noise. This makes the successful detection of an already fragile primitive even harder.

Controlling the level of noise in images was possible because the author was responsible for scanning the entries in the corpus, and consequently noise was kept to a minimum. Should an OMR system be intentionally aimed at processing poor quality images, then it is recommended that not just noise filtering techniques be used, but a suite of image restoration techniques be placed under the control of the operator [Cas96].

Chapter 6

Primitive detection

Classification of musical features is simplified by decomposing the problem into two steps. Step one detects the primitives shapes that make up music notation, such as note heads, stems, and tails. Step two assembles the detected primitive shapes into their respective musical features. This paradigm is often used in document image analysis problems [BBY92]. In this chapter we look at how to detect primitive shapes.

First we review pattern recognition techniques. The topic is vast, so emphasis is given to methodologies that are appropriate for the detection of musical primitive shapes. Next we present how existing OMR systems have used these techniques. This leads to a discussion of the options available, and the restrictions necessary for the design of an OMR stage that is capable of recognising an extensible set of primitive shapes.

In this discussion, we make the point that many of the techniques used by OMR systems to speed up recognition also compromise matching reliability. For instance, a heavy reliance on bounding box checks to narrow the possible set of shapes that an object might be, is intolerant of objects that are fragmented or touch accidentally. The use of musical context also results in fast recognition, but this is at the expense of robustness, since a mistake early on will quickly lead a recognition strategy astray. In a system where the areas checked for note heads are based on the located note stems, a misclassified note stem means no attempt is made to recognise its note heads. Alternatively, if the entire page is methodically searched for note heads then it is possible for the system to deduce the existence of the misclassified note stem, and thus correctly identify the musical feature, but this will take longer.

An OMR system, therefore, that provides access to a collection of pattern recognition techniques, whilst being neutral in enforcing a specific recognition strategy, is proposed as a versatile framework. For our work, this must be performed in an extensible manner. To meet this requirement, a specially designed programming language, called PRIMELA,[1] was developed. A PRIMELA description for a primitive specifies a set of distinctive features, and controls how the matching process proceeds. The remainder of this chapter details the design of PRIMELA, and evaluates its performance.

6.1 Pattern recognition techniques

The topic of pattern recognition is vast, and covers a wide variety of problems [CPW93]. Numerous techniques have been developed. This section describes ten methodologies that are appropriate for the detection of musical primitive shapes.

6.1.1 Template matching

Template matching is a general term that covers a variety of algorithms. In the standard algorithm, an idealised bitmap of an object, such as a filled-in note head, is compared with an unknown image and the number of pixels that match is recorded [BT88]. From this, a percentage proportional to a perfect match is computed, and used as a metric to either accept or reject the unknown shape. Variations include a *weighted exclusive-OR* match, and a closer study of the type of pixel mismatch [WMB94]. In the former variation, the importance of a mismatched pixel is influenced by the correctness of its neighbouring pixels. In the latter variation, pixel mismatches are separated into those that should be black but were found to be white, and those that should be white but were found to be black. This information is then used to calculate a more accurate match.

6.1.2 The Hough transform

The Hough transform is a method for evaluating the likelihood that a line described by a parametric equation exists at a specific place in the image [BT88]. For example, using a parametric equation that describes a vertical line, the Hough transform of a page of music indicates where note stems and bar lines are most likely.

[1] PRIMELA stands for PRIMitive Expression LAnguage.

Being a transform, the method works in a parameter space. The parameters in question are those used to describe the feature we are looking for. For pattern recognition purposes, our interest is in the two-dimensional case, where we shall use the representative parameters a and b to explain the method.

For each co-ordinate in the parameter space (a, b) we record the number of solutions that satisfy:

$$(a, b) = (x + f(t), y + g(t)) \tag{6.1}$$

where:

x and y are co-ordinates in the bitmap; and

$f(t)$ and $g(t)$ are the parametric version of the feature sought.

The value reflects the likelihood of the parameterised feature existing at this point.

The Hough transform is well suited for the identification of musical primitives, since it is insensitive to fragmentation and touching. However, depending on the transformation, the technique can be computationally expensive.

6.1.3 Feature classification

Feature classification is a general term used for functions that compute a particular property about an unknown shape, such as its centre of mass, area, or perimeter [CH67]. The technique is normally used when the set of possible shapes is small, and a feature exists that disambiguates between them. For example, calculating the area of an unknown shape will distinguish filled-in note heads from hollow note heads.

For larger sets, more than one feature is used to determine an unknown shape. The extension is known as "nearest neighbour."

6.1.4 Nearest neighbour classification

The nearest neighbour technique [CH67] works in parameterised n-dimensional Euclidean space. A collection of features are chosen that characterise the different shapes, with each feature forming an axis in n-dimensional space. For each type of shape that is to be classified, an idealised version is provided, and the values of its features computed. The result

is stored as a co-ordinate. To classify an unknown shape its features are computed, then the Euclidean distances between this point and all the idealised points are calculated, and the closest one chosen as a match. The technique can be computationally expensive, but approximations exist that are faster.

6.1.5 Neural networks

Neural networks have found many applications in pattern recognition domains [AB67, Pao93]. The classic configuration for pattern recognition is one where the input signals to the network are the pixel intensities of the unknown image, and the output signals form a vector indicating the classification, where each individual output signal represents a specific pattern. Thus, when an unknown shape is presented to the network, a single output signal—corresponding to the matched pattern—has the value one, and all the rest are zero. A neural network must be trained, prior to use.

6.1.6 Projections

Instead of matching an idealised template against an unknown shape, a weaker but faster solution is to compare an idealised projection against the unknown shape's projection [Pav82]. A projection in this context is a mapping from two-dimensions into one-dimension that can be taken at any angle, although in practice horizontal and vertical projections are the most common.

The comparison of two projections can be thought of as a 1-dimensional template match of a gray-scale image. Each corresponding entry is compared, and the further away the unknown value is from the ideal value, the less perfect the match is.

6.1.7 Slicing techniques

The technique of slicing involves taking one or more cross-sections through an unknown object at predetermined positions [AB67]. The number of transitions from black to white and white to black are counted, and compared with a database of results for the expected shapes [AB67]. For example, a vertical slice through the centre of a hollow note head gives two black cross-sections, whereas a vertical slice through the centre of a filled-in note head gives only one. A variation built on top of this is to include how long the cross-sections

are, and in most applications, a brief transition lasting for only one pixel is interpreted as noise, and is therefore ignored.

Counting the number of transitions in a slice computes a *feature* of the shape, so it is really a form of feature classification, with the extension to more than one slice being a form of the nearest neighbour strategy. The method is described separately since it is common to find recognition routines that make exclusive use of slicing techniques, rather than the nearest neighbour approach, which can handle heterogeneous metrics.

6.1.8 Bounding box check

This technique is as simple as it sounds: to determine a match, the bounding box of an unknown shape is checked against a list of shapes whose dimensions are known. By comparing bounding box sizes it is easy to distinguish between, say, a minim rest and a crotchet rest. Small tolerances are often included to compensate for noise and variations in font. Distances are typically calculated in a normalised space to free the recognition process from a particular font size or scan resolution. Like the slicing technique, a bounding box check is really an example of a feature computation.

6.1.9 Connectivity analysis

In connectivity analysis, a start and an end point are specified, and the method determines if a path of black pixels exists between the two points. A more stringent version of this algorithm restricts the path that can be taken between the two points—for instance, is there a path through the object from (x_1, y_1) to (x_2, y_2), where $x_1 < x_2$ with x monotonically increasing? Thus, the start and end point of a slur and a beam satisfy this monotonic restriction, but not the curl of a bass clef.

Even with path restrictions, many different shapes will pass the test. Post-conditional checks, such as a bound on the variance in the height of the shape travelled through, can be used to tighten a match.

6.1.10 Mathematical morphology

Mathematical morphology is a set theoretic approach [Rus92] applicable to many fields of computer science. Pattern recognition work is performed by representing a bitmap as the set of co-ordinates in the image that are black, and applying transforms that filter the

shapes present. After applying a series of transformations, the shapes that remain correspond to the pattern being sought. Two key operations are:

$$Dilation \quad : A \oplus B \quad = \{c \,|\, a \in A, b \in B, c = a + b\}, \text{ and} \tag{6.2}$$

$$Erosion \quad : A \ominus B \quad = \{c \,|\, b \in B, c + b \in A\}, \tag{6.3}$$

where A, and B are subsets of n-dimensional Euclidean space.

In pattern recognition work, A is the image undergoing transformation, and B is referred to as the structuring element, which is typically much smaller than the image. Informally, to dilate an image, the origin of the structuring element is placed over every pixel in the image that is black. The area swept out by the structuring element is the dilated image. Erosion works in reverse—only black pixels in the image that allow the rest of the structuring element to cover black pixels, are kept. For example, consider a filled-in elliptical structuring element that is slightly smaller than a note head, and an image that displays both filled-in and hollow note heads. Eroding such an image with this structuring element results in the elimination of hollow note heads.

Two commonly composed operations in mathematical morphology are known as *opening* and *closing*:

$$Opening \quad : A \circ B \quad = (A \ominus B) \oplus B \tag{6.4}$$

$$Closing \quad : A \bullet B \quad = (A \oplus B) \ominus B \tag{6.5}$$

where A, and B are subsets of n-dimensional Euclidean space.

The open operation can be used to separate objects that share a few common pixels, and the close operation can be used to fuse together small breaks in an object.

To implement mathematical morphology for pattern recognition purposes it is not necessary to use set operators, as the more efficient bitwise, boolean operations can be used instead. This reduces computational time.

6.2 Pattern recognition techniques in existing OMR systems

In an OMR system, the choice of a particular pattern recognition strategy is strongly influenced by the decision to either remove or ignore staff lines. If staff lines are removed, entire musical objects are isolated, and thus bounding box checks offer an efficient way to reduce the number of possible shapes an unknown object can be. If staff lines are ignored then musical context—such as accidentals appearing in front of note heads—is normally used to guide the recognition process. Below we summarise some of the main strategies that have been proposed in the past.

6.2.1 Staff lines removed

Prerau [Pre75] observed that different musical features vary considerably in size: a shape's bounding box, therefore, is a useful distinguishing feature; nevertheless, it is not always enough to uniquely classify a shape, and additional pattern recognition techniques may be required. Bounding box tests form the basis of many OMR systems where staff lines are removed.

Bounding box first

The system designed by Clarke *et al.* [CBT88b] combines bounding box checks with slicing techniques to classify musical features. The system first uses a bounding box check to reduce the number of possible matches. Slicing techniques, which compute both the number of transitions and the length of each transition, are then used to finalise the decision.

In the case of chord or note cluster[2] detection, a customised, directed slicing algorithm is used [CBT89]. First, horizontal slices are taken to decide if the note heads are at the top or the bottom of the stem. Next, a vertical slice is taken through the note head centre to fix the positions of the note heads (this is done on both sides in the case of a note cluster) and further vertical slices, away from the note head centre, are taken to disambiguate between ledger lines and hollow note heads.

In more recent projects [SER89, Bai91] the same bounding-box-first strategy is used.

[2]The term note cluster is used by Clarke *et al.* to describe chords where note heads are printed on either side of a note stem.

However, there has been a widening of pattern recognition methods utilised to disambiguate shapes.

Other OMR systems that remove staff lines have adopted different strategies.

Bounding box combined with other features

An experimental system by Fujinaga [Fuj88] exclusively used projections to see how successful the method was at recognising musical features. A fast system with a high rate of success was achieved, although the permissible set of music notation was simple and small. In the conclusion to this work, Fujinaga stressed the importance of a musical notation "vocabulary" that was dynamically extensible.

A more advanced system by Fujinaga addresses this issue [FAPH91, Fuj92], achieving its extensibility through a learning module. Based on alternate horizontal and vertical projections, large objects are recursively subdivided into smaller components, until simpler "atomic" objects are reached. Although not identical to the primitive shapes of music notation, atomic objects are similar in idea. Classification of an atomic shape is achieved using a nearest neighbour metric. The feature vector used is based on width, height, area, and centre of mass.

Although Fujinaga's system departs from the model of bounding-box-first, followed by disambiguation checks, it still retains Prerau's observation that many objects are distinguished by their bounding box. The bounding box is encoded within the feature vector along with some other discriminating features. Recognition, therefore, uses the bounding box and other features simultaneously in its decision.

The change in pattern recognition strategy by Fujinaga was motivated by the need to have an extensible vocabulary. The price to pay, however, is a significant rise in computational time [FPA91].

Mathematical morphology

The system designed by Roth [Rot94] detects the different primitive shapes in isolation. That is to say, all the vertical lines are found, then all the note heads, and so on. There is no use of bounding box checks to refine a search.

Recognition is accomplished using mathematical morphology. Using various structured elements in a sequence of transformations, the image is filtered, and the exposed

primitives are located and removed. The decision to remove primitives means that later checks for primitives can be simpler, which normally equates to faster.

Although Roth reports that the use of mathematical morphology to detect primitives leads to fast recognition algorithms, he also states that implementing the filtering rules with limits and tolerances for each primitive in a traditional programming language is tedious and time consuming. The task, he says, would be greatly simplified by the development of a higher-level language, where such rules and tolerances could be expressed more naturally.

A comparable model based on mathematical morphology has been presented by Modayur *et al.* [MRHS95]. Similar filters are used, although the overall matching control is more sophisticated.

Object oriented

The pattern recognition strategy devised by Reed also focuses on the detection of primitive shapes [Ree95]. Taking an object oriented approach to the problem, an object is created for each type of primitive shape to be recognised, storing in it whatever information is deemed necessary to complete the task. Thus, any pattern recognition routine desired can be programmed into an object, and contextual information can be exploited, if desired. The system developed uses a mixture of pattern recognition techniques, including template matching, vertical run-length searches, and character profiles [Kim91].

A three layer neural network

In the system by Yadid *et al.* all shapes are classified using a neural network [YBD+92]. Based on the "Neocognitron" [BC90], the binary images of isolated shapes are fed into the implemented network, which uses a three layer model to achieve classification: layer one responds to primitive features; layer two responds to combinations of small components from layer one; and layer three responds to whole note patterns such as a quaver note, a minim note, and a treble clef.

A skeletal graph and neural network

At the heart of the system by Martin and Bellissant is a planar graph that represents the skeletal decomposition of an object [MB91]. Each isolated object is thinned, segmented,

pruned, and then approximated by polygons. Each key point in the skeleton is categorised as either an end point, a junction point, or a bending point. These points form the nodes of the graph, and the arcs are segments of the skeleton that join these points together.

A variety of pattern recognition techniques are used to classify musical features. Notes are classified first. This is accomplished by searching the skeletal graph for vertical lines that match the system's requirement of a note stem. Template matching is then used to detect both filled-in and hollow note heads, based on the location of the potential note stems. Finally, horizontal slices are used to detect tails, and vertical slices are used to detect beams.

After processing the isolated objects for notes, the remaining shapes are classified. This is accomplished by encoding the skeletal graph using binary variables, which are then fed to a multi-layered neural network.

Line adjacency graph matching

The system by Carter is based on a transformed Line Adjacency Graph (LAG), and is unlike any other OMR system [Car89]. It not surprising, therefore, to find its classification algorithm is substantially different.

The technique can be seen as a sophisticated slicing method. The method for generating a transformed LAG was described in Section 2.6. From the point of view of a pattern recognition routine based on slices, a LAG is formed from vertical slices that saturate the entire scanned page. These vertical slices are conveniently grouped together as nodes in a transformed LAG, based on proximity and similarity in height. To detect a specific shape, therefore, all a slicing based recognition routine needs to do is traverse a transformed LAG looking for a particular configuration of nodes.

6.2.2 Staff lines ignored

Classification strategies in systems that do not remove staff lines, but ignore them during matching, are characterised by a heavy reliance on musical context to aid recognition. In particular, note head detection is often performed early on in these systems, so this information can be exploited in the recognition of subsequent shapes, such as guiding where to look for accidentals and duration dots.

The Wabot-2 project makes extensive use of template matching early on in the classification process [MHS+85]. The key patterns recognised using this technique are filled-in note heads, bar lines, and accidentals.

With a better established awareness of the piece, based on the location of staves, bar lines and filled-in note heads, further classification is aided by musical context, which reduces the number of possible shapes at a given point. For example, it is assumed that the first object encountered on a staff will be a clef. A discrimination unit is then used to finalise classification. Because the number of possible shapes are fewer, simpler pattern recognition strategies can be employed. Slicing is the primary method used. To discriminate between a treble clef and a bass clef, the Wabot-2 project takes two horizontal slices: one between the second and third staff line (counting up from the bottom), the other between the fourth and fifth staff line. By measuring the length of the first white cross-section in both slices (in other words, the length from the start of the staff to the start of the clef), the unknown clef can be determined. Slicing techniques are also used in recognising hollow note heads and counting the number of tails or beams a note stem has.

The Wabot-2 project was the first significant work that chose to ignore staff lines, and has had much influence over later projects that chose the same route [LC85, Gla89, IIHO92, MRHS95]. Although the newer projects have developed more sophisticated control over the classification process, the same pattern recognition techniques are being used at the root of these algorithms.

6.3 Implications of existing work

Here we discuss the implications of existing work with regard to the design of a pattern recognition stage for an extensible OMR system. The goal is to keep as many options open as possible.

In the Wabot-2 project we have seen how the targeting of note heads early on in the system means that specific areas can be searched for tails, beams, and accidentals. The strategy, however, is not exclusive to OMR systems that ignore staff lines, as the same ideas can be used in systems that remove staff lines. For example, based on what the previous object was classified as, the author's prototype system [Bai91] reduces the list of possible matches that an unknown, isolated object can be. Also, within the recognition of a

note, the system targets areas to check for note heads, based on the location of note stems.

Exploiting musical context leads to fast recognition strategies, but this is at the expense of robustness and reliability. If the Wabot-2 system fails to identify some note heads, or the author's prototype system fails to identify some note stems then important areas that should be checked for primitive shapes are missed. Alternatively, an unguided system that makes use of the same pattern recognition techniques might take more time to detect the primitive shapes in the page, and it too would fail to classify the deformed note heads or stems. However, because all the surrounding primitive shapes are detected, the possibility exists for the system to deduce that a note exists at that point on the page. Therefore, in the design of our pattern recognition stage that is capable of classifying arbitrary primitive shapes, we do not want to *force* the use of musical context to process shapes. Different applications have different requirements with regard to speed and reliability. Consequently, the ability to exploit musical context should be an *option* in the system.

A popular strategy in OMR systems that remove staff lines is to first narrow the set of possible matches based on the unknown object's bounding box. Like systems that exploit context, this too is a strategy that results in fast recognition systems. However, the same trade-off occurs, since this strategy also weakens the robustness of the matching process. When an object becomes fragmented, the resultant bounding boxes are no longer a true reflection of the shape's dimensions. The same is true for objects that touch. Another factor to consider when using bounding boxes to classify shapes is that as the set of musical shapes to be recognised increases, so too does the task of designing algorithms that distinguish between the various shapes with similar dimensions.

Again, a requirement of our pattern recognition stage is not to force such a strategy for recognition, but allow it as an option, and the same is true for a strategy that removes primitives as they are detected. In fact, this statement can be broadened to any classification strategy and the pattern recognition techniques it uses.

All recognition methods have their strengths and weaknesses. A versatile approach in an extensible system, therefore, is a neutral recognition framework that provides access to a collection of pattern recognition techniques, and can be shaped to use a particular strategy if desired. The person responsible for configuring the system—who understands the requirements of the system—decides what knowledge the recognition strategy can exploit, and which pattern recognition routines should be used to recognise which primitives.

It was decided that a suitable framework to implement such a design would be a specially designed programming language, crafted to abstract the essence of the task, whilst cutting away much of the tedious, repetitive detail required to realise a specific OMR system.

6.4 The requirements of PRIMELA

PRIMELA is a language specifically designed to aid the task of classifying the primitive shapes found in music notation. In this section we study the requirements of the language.

To perform the classification of a primitive shape, we must specify enough distinguishing features, and detail how the resultant information is to be used, to determine a match. For example, one way to classify a treble clef might be to compute the number of black pixels in a shape. If the count lies between C_1 and C_2, then the shape is classified as a treble clef. To provide a general mechanism for this the programming language needs:

1. a collection of pattern recognition routines, and
2. control over the matching process.

6.4.1 Primitive categories

The primitive shapes used in music notation can be broadly categorised into three groups.

- *Text-like shapes* that vary little in size and are approximately the same size as regular text. Examples include accents and stress marks. They are usually isolated components.
- *Distinctive sized shapes* that vary little in size: examples include the treble clef, the bass clef, note heads, and rests.
- *Variable sized shapes.* Examples include hairpin crescendos/diminuendos, slurs, beams, and note stems.

The groupings characterise the type of pattern recognition routines that are applicable. *Text-like shapes* can be recognised using techniques developed for OCR. *Distinctive sized shapes* already contain the information necessary for classification, i.e. their size.

Classification based on their bounding box will quickly reduce possibilities, and final classification can be accomplished with a follow up check, such as a slicing technique or a projection match. *Variable sized shapes* are not so common in other pattern recognition problems, and thus require techniques that support parameterised feature descriptions, such as the Hough transform, slicing techniques and connectivity analysis.

A programming language that recognises the primitive shapes of music notation will require a collection of pattern recognition routines that covers these categories.

6.4.2 Arbitrary graphical shapes

Some pattern recognition techniques compute predefined quantities, such as perimeter, area, width, and height. Such algorithms are simple to call, and therefore easy to include in a programming language. Other recognition techniques, however, rely on specialised graphical information which is provided when the routine is invoked. For example, to determine if an unknown object includes a particular primitive, a template matching algorithm needs an idealised version of the shape upon which to base its comparison. Another requirement of the programming language, therefore, is the ability to specify arbitrary graphical shapes.

6.4.3 Scale independence

Another attribute the programming language must address is scale independence. Since there is no standard size for music notation, quantifying distinguishing features at a particular resolution is of little use. Such information is required in a scale independent form, including any arbitrary graphical shapes. Specified shapes, therefore, must be given in a normalised space *and* be scalable to any size without loss of detail.

6.5 A sample PRIMELA description

Before we formally explain the design of PRIMELA, a sample PRIMELA description is given to introduce the reader to the syntax and structure of the language. Shown in Figure 6.1, the listing describes how a filled-in note head is recognised.

First the name of the primitive is specified (line 1) along with some properties of the canvas that will be used to draw the idealised version of the note head (line 2). The re-

```
 1 primitive full_note_head
 2  : size(31,21), origin(15,10), normalise(1.0)
 3 {
 4   template tfnh
 5     {
 6       transform: rotate(2)
 7         {
 8           ellipse(1) origin(15,10) axis(31,21);
 9           floodfill(15,10,1);
10         }
11     }
12
13   initialise
14     {
15       int option_box  = "object";
16
17       int tplate_lower_match  = 75;
18       int tplate_upper_match  = 78;
19
20       int x_copy_expand         = 1;
21       int y_copy_expand         = 1;
22     }
23
24   match
25     {
26       pre       {% true %}
27
28       rect      {% rect_xl_ -= x_copy_expand;
29                    rect_xr_ += x_copy_expand;
30                    rect_yt_ -= y_copy_expand;
31                    rect_yb_ += y_copy_expand;              %}
32
33       // implicit match
34
35       copy      {% %}
36       post      {% true %}
37
38       certainty {%  linear_certainty(tplate_lower_match,
39                                        score_match_,
40                                        tplate_upper_match); %}
41
42       remove    {% true %}
43       noise     {% false %}
44     }
45
46 }
```

Figure 6.1: The PRIMELA description for a filled-in note head.

mainder of the description is divided into three sections: pre-specified template (lines 4–11), variable initialisation (lines 13–22), and execution control of the matching process (lines 24–44).

A pre-specified template takes the form of a nested block structure, headed by the type of pattern recognition routine to use—template matching in this case (line 4). Basic commands that draw simple shapes and fill in an enclosed region are combined into nested blocks, where a geometric transform involving translation, scaling, and rotation

can be applied. In this example, a filled-in note head template is generated by drawing an ellipse (line 8) which is then filled-in (line 9) and finally rotated by 2° (line 6).

Variables included in the initialisation section are either predefined variables that control the search for primitives, or dynamic variables introduced by the configurer[3] to aid the matching process. The entire image could be searched for a primitive, but faster options are available through the predefined `option` variables, although the match performed is not necessarily as reliable. By setting `option_box` to `"object"` in the example (line 15), the search for note heads is restricted to the rectangular areas defined by the bounding boxes of the isolated shapes. The dynamic variables `tplate_lower_match` and `tplate_upper_match` set a two tier threshold for an acceptable match: a match below `tplate_lower_match` is definitely *not* a match; a match above `tplate_upper_match` definitely *is* a match; and a match between these two values is a *possible* match. The last two variables, `x_copy_expand` and `y_copy_expand`, are used to adjust the size of the rectangle searched in an attempt to combat the effects of noise.

Control of the matching process forms the last section of a description and is decomposed into seven steps: `pre`, `rect`, `copy`, `post`, `certainty`, `remove`, and `noise`. Using these steps, a description specifies the complexity of a match, which can vary from simplistic to sophisticated.

The first step is a pre-conditional check that must be passed before an application of the pattern recognition routine is considered. A common check is to discount objects whose bounding box is too large. The second step provides an opportunity to alter the rectangular area that is checked. A common use of this is to enlarge the area slightly to compensate for noise, as is done in the example (lines 28–31). At this point the pattern recognition routine is implicitly invoked and potential matches identified. For each potential match, a copy of the detected primitive is taken, and a post-conditional check made. Some pattern recognition techniques, such as connectivity analysis, are generous in what shapes pass the test. The post-conditional check can be used to tighten the requirements of a match by insisting, for example, that the detected shape is at least 60% black. Uncertain data is supported in the language by assigning a value between zero and one to the detected primitive, where a value of one represents a definite match. Only a definitive match is removed

[3]Recall the configurer is responsible for adapting the extensible OMR system to meet the end-user's needs.

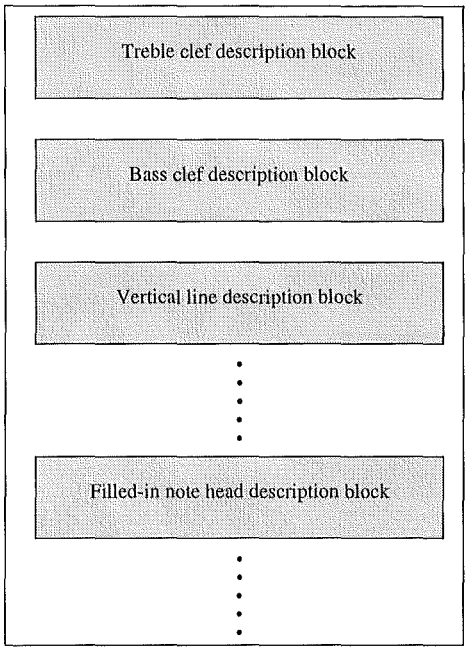


Figure 6.2: In PRIMELA, the set of primitives to be recognised is entered as a sequence of descriptive blocks.

from the image. Finally the area that has been checked for primitives is filtered for noise.

When a step is left empty (line 35) a default action is substituted. In the case of `copy` the default action is to copy the entire rectangular region covered by the template. More sophisticated functions can be invoked if necessary.

Currently the configurer is required to type in PRIMELA descriptions. In the future this could be replaced with a graphical user interface, thus reducing the computer programming expertise required.

6.6 The design of PRIMELA

To specify all the primitive shapes to be recognised, a set of primitive descriptions is required. In PRIMELA this is accomplished by making the description for an individual primitive a separate block, which is then sequenced with other blocks. A schematic example of this is shown in Figure 6.2. The idea is similar to standard programming languages, where a program is laid out as a sequence of procedures.

Each primitive description block specifies how a particular primitive shape is detected. Here, the necessary features to distinguish a primitive, and the control of the

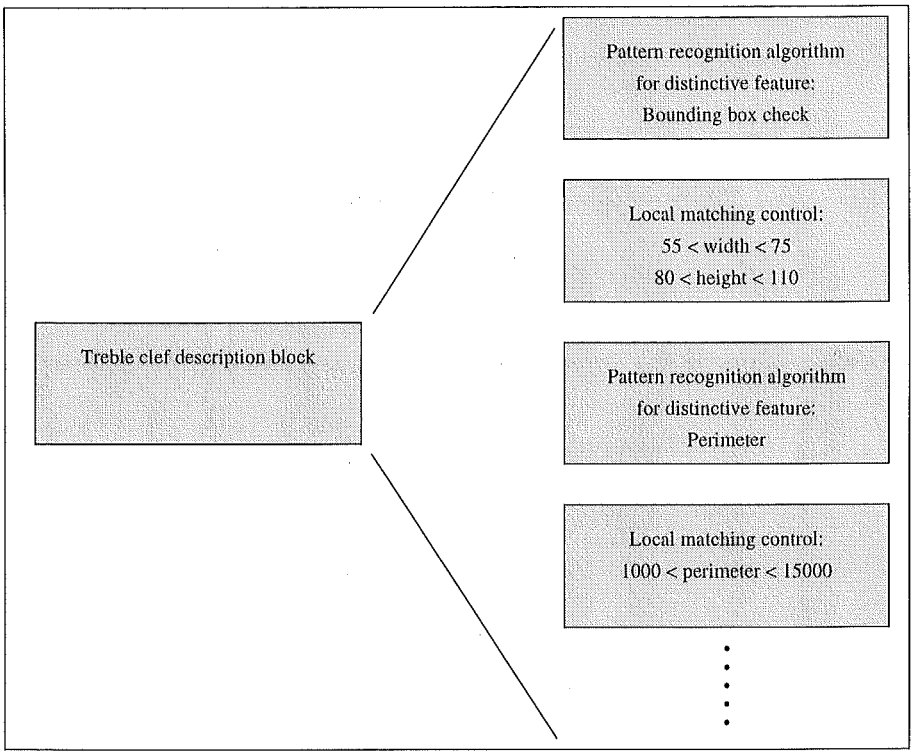


Figure 6.3: A PRIMELA primitive description block is decomposed into pairs of pattern recognition matching algorithm and local control of the matching process.

matching process, are detailed together. For each distinctive feature within a primitive description, the pattern recognition algorithm to use is defined, along with local control of the matching process. A schematic example of this is shown in Figure 6.3. To be classified as a particular primitive, an object must pass the tests for all the distinctive features.

We will now consider in more detail the constructs necessary to invoke a pattern recognition routine that detects a distinctive feature, and how to structure the local matching control process so as to offer different strategies as options.

6.6.1 Invoking a pattern recognition routine

To cover the different categories of musical primitive shapes (Section 6.4.1) the pattern recognition techniques accessible from PRIMELA are: template matching, bounding box check, projections (vertical and horizontal), Hough transform, slicing techniques, and connectivity analysis. There is also the provision for new pattern recognition algorithms supplied by the configurer. Table 6.1 shows which techniques are applicable to which primi-

	Text-like	Distinctive size	Variable sized
Template matching	X	X	
Bounding box check		X	X
Projection	X	X	
Hough transform	X	X	
Slicing techniques	X	X	X
Connectivity analysis			X
Configurer supplied	X	X	X

Table 6.1: How the selected pattern recognition techniques for PRIMELA cover the primitive categories.

tive categories. To invoke one of these routines, a PRIMELA description specifies the name of the desired recognition routine at the start of the distinctive feature block.

6.6.2 Arbitrary graphical shapes

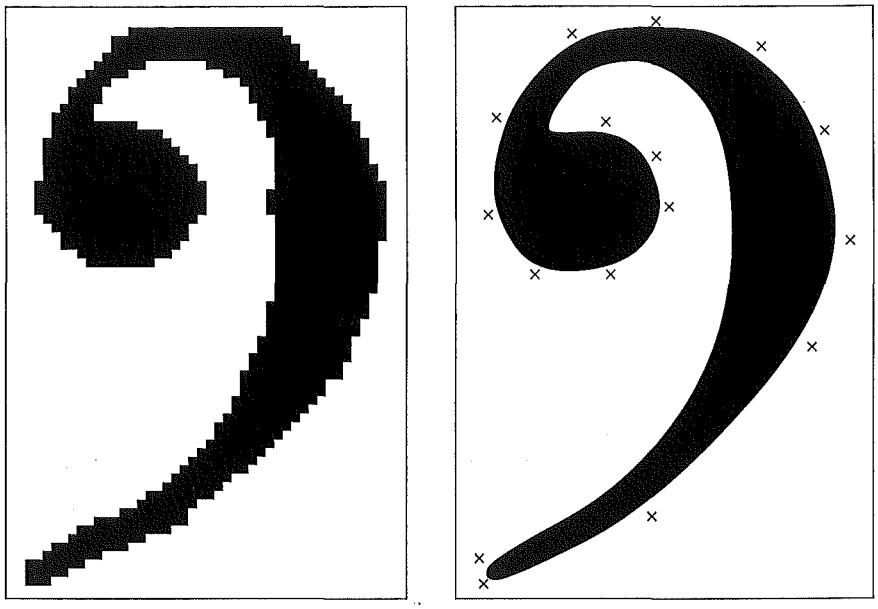
Some of the recognition algorithms need a graphical shape to be defined. Figure 6.4 shows two possible solutions.

A bitmap editor (Figure 6.4a) facilitates the drawing of arbitrary graphical shapes. Unfortunately, the level of detail provided is fixed. A scaled down version of the bitmap loses detail, and an enlarged bitmap becomes "chunkier." A bitmap editor, therefore, does not meet our requirement of scale independence.

In contrast, a drawing package (Figure 6.4b) allows the user to create arbitrary graphical shapes that are scale independent. Armed with a suite of geometric shapes, the user can draw objects any size, with any line thickness, and with a choice of patterns, shades and colour for the outline or interior of the object. This can then be scaled to any size, without loss of detail.

Our task of drawing graphical shapes for pattern recognition routines does not require the comprehensive mechanism for constructing arbitrary graphical shapes found in commercial drawing packages. We only need a conservative (yet equally powerful) suite of shapes, drawn at unit thickness, that can be coloured in with shades of gray, where the intensity of gray-level used determines the relative importance of the area in the matching process.

To recognise a treble clef using a horizontal projection, an idealised version of the primitive is extracted from the scanned image, cleaned up using a bitmap editor, and its horizontal projection taken (Figure 6.5). This horizontal projection is then specified as the



(a) (b)

Figure 6.4: Drawing arbitrary graphics (a) a bitmap editor (b) a drawing package.

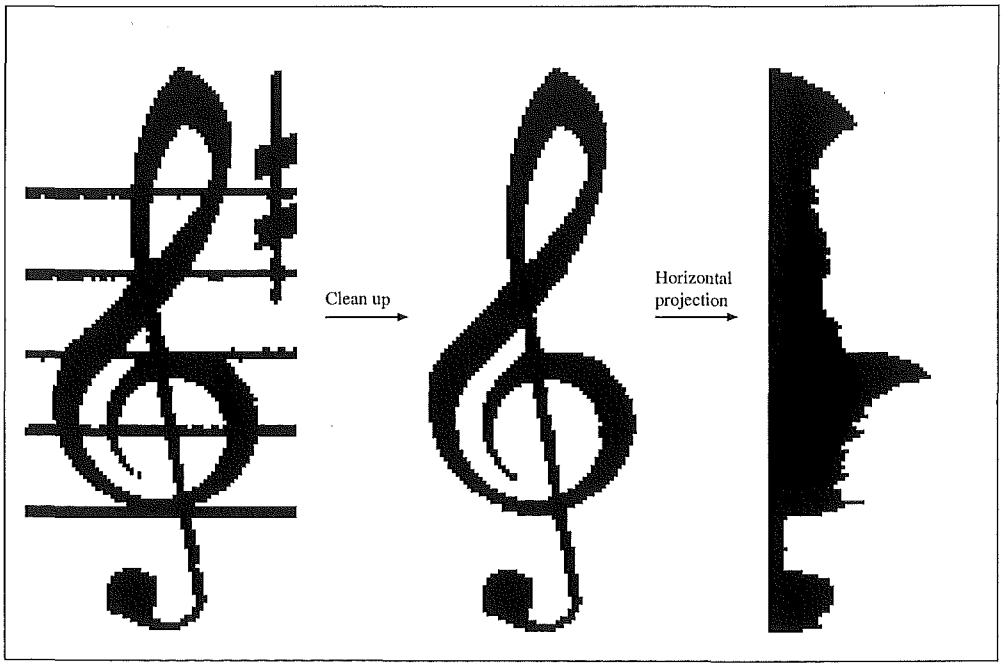
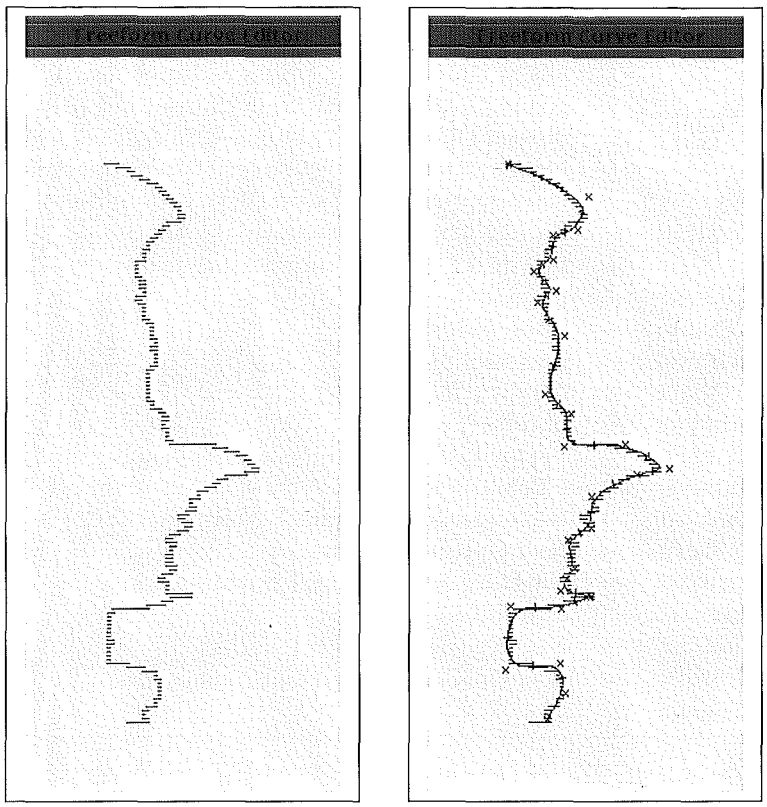


Figure 6.5: Extracting and processing a treble clef for the customised drawing package.



(a) (b)

```
1  transform: translate(75,310), scale(0.25,0.25)
2    {
3      bspline 2 open 28
4          (-73  108) (-12   92) (-20   66) (-39   62) (-39   43) (-54   34)
5          (-37   19) (-51   10) (-30  -16) (-45  -61) (-25  -79) (-30 -101)
6          ( 17  -99) ( 51 -118) ( 26 -122) (-15 -139) ( -9 -163) (-18 -172)
7          (-22 -104) (-33 -211) ( -9 -216) (-15 -216) (-32 -225) (-71 -223)
8          (-75 -272) (-33 -267) (-29 -290) (-43 -310);
9    }
```

(c)

Figure 6.6: Generating the PRIMELA code for a graphical shape (a) the backdrop to the drawing package (b) the fitted line (c) the automatically generated PRIMELA code.

backdrop for the customised drawing package, to be viewed at a magnification factor of four (Figure 6.6a). The configurer then fits a line to the displayed shape, manipulating the control points until satisfied that the lines capture the essence of the shape (Figure 6.6b), at which point the picture is saved to a file as a fragment of PRIMELA code (Figure 6.6c).

1	Graphic	→	Freeform ; | FloodFill ;
2	Freeform	→	**line** OptGrayLevel FState NumPoints BsplineList
3		|	**circle** OptGrayLevel Origin **radius** (*float*)
4		|	**circle** OptGrayLevel Origin **diameter** (*float*)
5		|	**ellipse** OptGrayLevel Origin **axis** (*float* , *float*)
6		|	**bspline** OptGrayLevel Order FState NumPoints BsplineList
7		|	**nurb** OptGrayLevel Order FState NumPoints NurbList
8	FloodFill	→	**floodfill** (*integer* , *integer*)
9		|	**floodfill** (*integer* , *integer* , *integer*)
10	Order	→	*natural*
11	FState	→	**open** | **closed**
12	OptGrayLevel	→	(*natural*) | ϵ
13	Origin	→	**origin** (*float* , *float*)
14	NumPoints	→	*natural*
15	BsplineList	→	BsplineList BsplineInterval | BsplineInterval
16	BsplineInterval	→	(BsplineKnotList)
17	BsplineKnotList	→	BsplineKnotList BsplinePt | BsplinePt
18	BsplinePt	→	*float* OptComma *float*
19	NurbList	→	NurbList NurbInterval | NurbInterval
20	NurbInterval	→	(NurbKnotList)
21	NurbKnotList	→	NurbKnotList NurbPt | NurbPt
22	NurbPt	→	*float* OptComma *float* OptComma *float*
23	OptComma	→	, | ϵ

Figure 6.7: The syntax for arbitrary graphical shapes in PRIMELA.

Figure 6.7 details the syntax for the possible shapes that can be specified in the PRIMELA language. The syntax is expressed as a grammar: terminal symbols in the language are written in **bold**, and placeholders for different types of numbers are expressed in *italics*.

Lines 2–7 in the grammar represent the available shapes: **line**, **circle**, **ellipse**, **bspline**, and **nurb**. B-splines represent a powerful category of geometric shape, forming a super-set of curves, including hyperbolas, parabolas and ellipses [FvDFH90]. It is a method that unifies all geometric shapes, being able to draw straight lines through to irregular curves. There are, however, simpler ways to draw regular shapes such as circles and rectangles than using a B-spline. In a normal drawing package, therefore, regular shapes are accessible directly, and B-splines are only intended for drawing irregular curves, and thus appear in a restricted form, requiring only the $x - y$ co-ordinates of the control points to be specified. The **bspline** construct in PRIMELA follows this use, where the non-terminal symbol BsplinePt on line 18 specifies a control point to be two floating-point numbers. To draw elliptical arcs, hyperbolas and so on, the unrestricted form of B-

1	GraphicBlocks	→	GraphicBlock GraphicBlocks | ϵ
2	GraphicBlock	→	TransformBlock | Graphic ;
3	TransformBlock	→	**transform** : OptTransformList { GraphicBlocks }
4	OptTransformList	→	Trans , OptTransformList | ϵ
5	Trans	→	**translate** (*float* , *float*)
6		|	**rotate** (*float*)
7		|	**scale** (*float* , *float*)

Where Graphic *is the grammar symbol defined in Figure 6.7.*

Figure 6.8: The PRIMELA syntax for sequencing and nesting shapes.

splines—known as nonuniform rational B-splines, or NURBS—must be used [FvDFH90]. This is the reason for including the **nurb** construct in PRIMELA, which requires three floating-point numbers per control point: the x co-ordinate, the y co-ordinate, and the *weight* component, as specified by the non-terminal symbol NurbPt on line 22.

Lines 8–9 in the grammar represent the construct **floodfill**. The effect of **floodfill**(x,y,gl) is to fill in the outlined shape that encloses the co-ordinate (x,y) with the gray-level gl. The different levels of gray are used to specify the relative importance of a region in a match. The higher the value, the more important the region is. If no gray-level is specified, the value one is used.

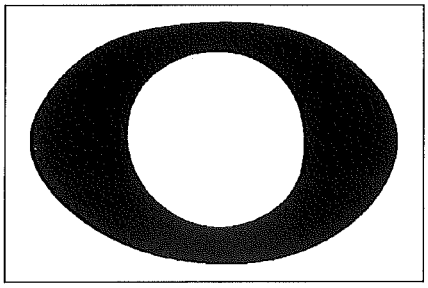
Drawn shapes can be sequenced, and sequences of shapes can be grouped together as a transformation block, and subjected to a translation, a scaling, and a rotation. A transformation block itself is treated as a single entity, and hence can also be sequenced or nested. The syntax for both of these constructs is shown in Figure 6.8.

Transformation blocks are purely for the convenience of the configurer. There is no nested structure of shapes that cannot be represented as a flat sequence of shapes, but it is sometime more convenient for a configurer to describe a composite shape in a nested manner.

Figure 6.9a shows the graphic block of a PRIMELA description for a semibreve note head: lines 4–6 specify an outer ring, lines 9–10 specify an inner ring, and line 12 specifies a point between the two rings which is used as the seed for a flood-fill operation. The transform block (line 1) translates the resultant shape so it is placed centrally on the canvas, which is then converted to the normalised space through a scaling operation. Figure 6.9b shows the corresponding image.

```
1  transform: translate(173,107), scale(0.1)
2    {
3      // outer
4      bspline(1) 3 closed 12
5        (-173 1) (-149  53) (-95 102) (-4  113) ( 80 109) ( 149  60)
6        ( 173 3) ( 143 -58) ( 79 -98) ( 9 -107) (-71 -95) (-139 -53);
7
8      // inner
9      bspline(1) 3 closed 8
10       (-83 2) (-69 64) (2 95) (70 66) (86 2) (71 -52) (5 -80) (-56 -52);
11
12     floodfill(-100,0,1);
13   }
14 }
```

(a)



(b)

Figure 6.9: Specifying the graphical shape for a semibreve note head in PRIMELA (a) the PRIMELA description (b) the corresponding graphical representation.

6.6.3 Local matching control

Matching control is separated into: how and where to search for shapes within the image; and what to do once you have decided to apply the pattern recognition routine to a particular spot. Both mechanisms offer options that vary the robustness and computational cost of the match.

As an example, the application of the Hough transform with the parametric curve shown in Figure 6.10 detects bass clef curls. Checking every position in the image is reliable, but expensive computationally. One option is to only apply the recognition algorithm if the position being checked starts with a black pixel. In PRIMELA this is accomplished by setting the variable `option_pixel` to `"black"` (Figure 6.11a). Although this check is faster, the test is less reliable since bass clef curls in an image are not ideal, and

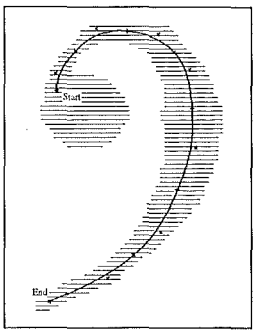


Figure 6.10: The parametric curve used by the Hough transform to detect bass clef curls.

```
1  initialise
2    {
3      // option settings
4      string option_pixel          = "black";
5
6      // match settings
7      int hough_lower_match = 75;
8      int hough_upper_match = 80;
9    }
```

(a)

```
1  initialise
2    {
3      // option settings
4      string option_box          = "object";
5
6      // match settings
7      int hough_lower_match = 75;
8      int hough_upper_match = 80;
9    }
```

(b)

```
1  initialise
2    {
3      // option settings
4      string option_box            = "object";
5      int    option_extend_percent = 150;
6
7      // match settings
8      int hough_lower_match = 75;
9      int hough_upper_match = 80;
10   }
```

(c)

Figure 6.11: Controlling how to search for a primitive (a) only check positions that start with a black pixel (b) only check areas defined by the bounding boxes of isolated objects (c) only check areas defined by the extended bounding boxes of isolated objects.

Variable name	Description
`option_box`	Controls whether the entire image or only the isolated objects are searched.
`option_pixel`	Controls when a pattern recognition routine is invoked. This can be when the origin pixel under consideration is black, white or either.
`option_x_extend_percent`	Specifies the amount a rectangular region is enlarged along the x-axis.
`option_y_extend_percent`	Specifies the amount a rectangular region is enlarged along the y-axis.
`option_extend_percent`	Specifies the amount a rectangular region is enlarged along both axes.
`option_extend_dir`	Controls which directions are enlarged: up, down, left and right.
`option_wbitmap`	Controls which image is passed to the recognition routine.

Table 6.2: Predefined variables in PRIMELA that control searching.

a defective bass clef curl that has lost its starting portion on the F staff line, will be excluded from the search. Fortunately this particular scenario is unlikely, and the faster test yields high accuracy.

Another option is to restrict the search to rectangular areas based on the bounding boxes of the isolated shapes, reducing the computational cost for many primitives. In PRIMELA this is accomplished by setting the variable `option_box` to `"object"` (Figure 6.11b). Problems arise when the primitive being searched for is fragmented, since the bounding boxes of the fragmented object may be too small to contain the parametric equation, and consequently be excluded from the match. Increased reliability is available in PRIMELA through the variable `option_extend_percent`, which is used to extend the dimensions of the rectangular area searched (Figure 6.11c). See Appendix B for more details.

Table 6.2 summarises the range of searching options. The ability to extend rectangular regions is intended for use with isolated objects. It makes no sense to enlarge the dimensions on the entire image, since all the pixels are contained in the rectangular region already.

When *searching* for a place to apply the recognition routine for a primitive, the image used is one where the staff lines have been removed. However, to *perform* the match, this is not the only image that could be used—the original scanned image is also a useful location. If fragmentation of a primitive is likely, the original image might be a more re-

PRIMELA keyword	Description
`pre`	A pre-condition for matching can be specified.
`rect`	In compensating for noise, being able to alter the rectangle to be searched is useful.
`copy`	Once a match has been found, a copy of it is made. The copying procedure can be supplied by the configurer.
`post`	A post-condition for matching can be specified.
`certainty`	Uncertainty is supported in the recognition process by assigning a "certainty rating" to each primitive detected. This step can be used to control the assigned value.
`remove`	A detected shape with a certainty rating greater than or equal to one is removed. The removal procedure can be supplied by the configurer.
`noise`	An image can be filtered for noise. The function used can be specified by the configurer.

Table 6.3: The seven steps that control the execution of a PRIMELA match.

liable place to perform the match. In PRIMELA, the location to perform a match is controlled by the predefined variable `option_wbitmap`.

Controlling the match process

Once it has been decided that a particular spot in the image is worth checking for a primitive, the matching process commences. The basic task is to return a rating that indicates how similar the given spot is to the desired primitive, and using this information to decide if the primitive exists there.

It is possible for a matching process to consist only of this basic task. However, as we have seen earlier in this chapter, more complicated matching processes have been proposed in the literature. For example, in the system by Roth [Rot94], matched primitive shapes were also removed from the image.

In PRIMELA, the execution of a match consists of seven steps. An ordered list of the steps is shown in Table 6.3, where a description of each step and the PRIMELA keyword used, is given.

All seven steps are optional. If none are used, then the matching process degenerates into the simple strategy of returning a rating that indicates how similar a particular spot is to a particular primitive. However, pre-conditions and post-conditions can be used to tighten the requirements of a match, or lower the computational cost; removing a

Construct	Type
`pre`	boolean
`rect`	void
`copy`	void function(integer x, integer y)
`post`	boolean
`certainty`	floating point
`remove`	void function(integer x, integer y)
`noise`	boolean

Table 6.4: The required expression types for matching control constructs.

detected primitive can simplify the matching requirements of later primitives, and simultaneously lower the computational cost of detecting subsequent primitives types since it reduces the number of interesting places that need to be checked; and finally, computing certainty ratings can be used to improve the reliability of the primitive assembly process.

The **initialise** block is not only for the predefined searching variables. New variables declared here can be accessed during the matching control.

6.6.4 Detecting a single feature

In the preceding sections we have compartmentalised the detection of a single feature: after specifying a particular pattern recognition routine, an optional graphical description of the shape is provided; then, through the use of predefined variables, we specify how to search for the primitive feature, and where the match should take place; and finally we detail the execution of the match. Figure 6.12 shows a schematic version of the structure. Figure 6.13 gives the PRIMELA syntax that binds these compartments together.

As we shall see later, some parts of a PRIMELA description are translated into a traditional programming language. It is therefore convenient in the language design to allow fragments of the chosen target language to be embedded, rather than reinventing many widely used programming constructs. This is done using the {% *raw text* %} construct in Figure 6.13 (lines 15–21).

In terms of the parser, any characters can appear between the special brackets {%...%}, but to function correctly the text must be valid syntax for the target programming language. In fact, for a valid PRIMELA description there are stricter requirements that these sections of "raw text" must meet: all must be expressions. The type of the expression that can be used for each construct is shown in Table 6.4.

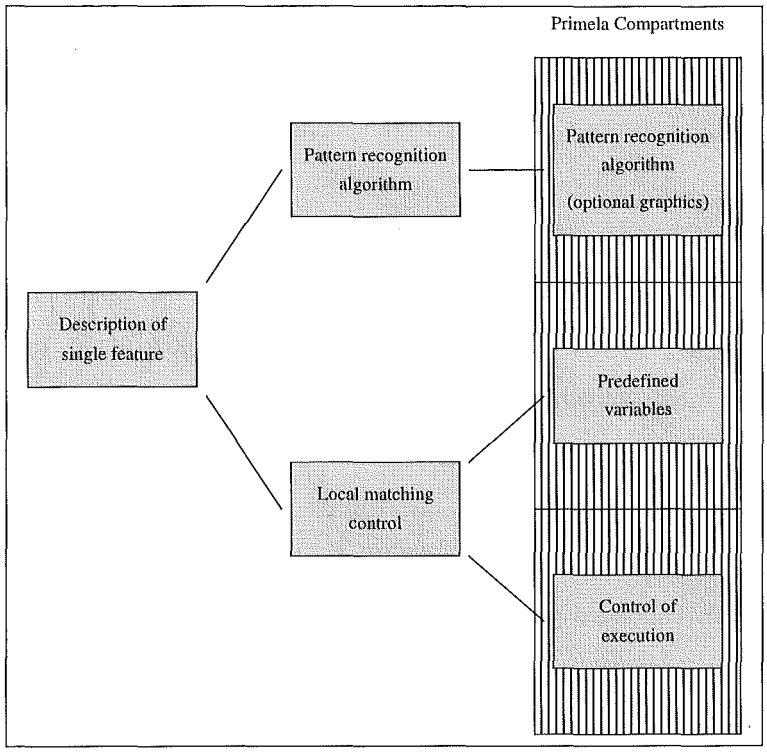


Figure 6.12: How a single feature is compartmentalised in PRIMELA.

1	PrimDescBlock	→	PrTypeBlock InitBlock MatControlBlock
2	PrTypeBlock	→	PrType *identifier* { GraphicBlocks }
3		|	**bbox** *identifier* { }
4	PrType	→	**template** | **hough** | **slice**
5		|	**connect** | **xproject** | **yproject**
6		|	**user** *identifier*
7	InitBlock	→	**initialise** { VarDecs }
8	VarDecs	→	VarDec VarDecs | ϵ
9	VarDec	→	**int** *identifier* = *integer* ;
10		|	**float** *identifier* = *float* ;
11		|	**string** *identifier* = *string* ;
12	MatControlBlock	→	**match** { MatControlItems }
13	MatControlItems	→	OptPre OptRect OptCopy OptPost OptCert
14			OptRemove OptNoise
15	OptPre	→	**pre** {% *raw text* %} | ϵ
16	OptRect	→	**rect** {% *raw text* %} | ϵ
17	OptCopy	→	**copy** {% *raw text* %} | ϵ
18	OptPost	→	**post** {% *raw text* %} | ϵ
19	OptCert	→	**certainty** {% *raw text* %}
20	OptRemove	→	**remove** {% *raw text* %} | ϵ
21	OptNoise	→	**noise** {% *raw text* %} | ϵ

Where GraphicBlock *is the grammar symbol defined in Figure 6.8.*

Figure 6.13: The PRIMELA syntax to describe a single primitive feature.

6.6.5 Combining single features to detect a primitive

The final step in building a complete description for a primitive is to sequence the single features together, and apply a wrapper that specifies top level information, such as the name of the primitive. An example description for a treble clef is shown in Figure 6.14. The keyword **primitive** starts a description, followed by its name. Next, information is specified that determines the size of a primitive with respect to the height of the normalised staff height.[4] The drawing of graphical shapes takes place on some arbitrary grid—whatever is most convenient to the configurer. The scale factor required to map this grid into the normalised co-ordinate space is given by the **normalise** construct on line 2. Also specified on this line is the unnormalised rectangular size of the canvas used for the drawing operations and the unnormalised origin to be used during the match. The PRIMELA syntax for this final part of the language is given in Figure 6.15.

The **functions** construct on line 2 of Figure 6.15 is a mechanism for configurer supplied functions. Typical functions are new pattern recognition algorithms, and supporting functions used in the execution control of a single feature match.

6.7 The implementation of PRIMELA

In PRIMELA, some parts of the language are naturally suited to a translator approach, whilst other parts are better suited to be interpreted. PRIMELA, therefore, was implemented as a hybrid translator/interpreter strategy. Compiler tools greatly simplified the task. Using a lexical analyser with a parser generator, the developed software accepts a PRIMELA program as input, and produces a linked list of tree data-structures that represent the program. This can be traversed to generate the translated code, or processed by the interpreter.

Since the interpreter performs image processing work, the imperative programming language C++ was chosen as the target language for the translator. For further implementation details see Appendix B.

[4]The normalised staff height was somewhat arbitrarily set at 86. This is the height of a standard staff rendered at 300 dpi by the musical notation typographical system MusicTEX, which was the initial resource the author used to measure the dimensions of musical features.

```
 1 primitive treble_clef
 2  : size(42,145), origin(21,40), normalise(1.0)
 3 {
 4   yproject ptc
 5     {
 6       transform: scale(0.333333,0.333333), translate(23.333333,106.333333)
 7         {
 8           bspline 2 open 14
 9                     (-70  115) (-01   95) (-02   62) (-61   22) (-48  -39)
10                     (-27 -100) ( 56 -107) ( 43 -133) (-44 -175) (-09 -217)
11                     (-65 -230) (-67 -271) (-29 -293) (-56 -319);
12         }
13     }
14
15   initialise
16     {
17       string option_box = "object";
18
19       int yproject_lower_match = 70;
20       int yproject_upper_match = 75;
21
22       int y_copy_expand  = 2;
23     }
24
25   match
26     {
27       pre       {% %}
28       rect      {% rect_yb_ += y_copy_expand;                          %}
29       copy      {% %}
30       post      {% %}
31       certainty {% linear_certainty(yproject_lower_match,score_match_,
32                                     yproject_upper_match);             %}
33       remove    {% true %}
34     }
35 }
```

Figure 6.14: An example PRIMELA description for the treble clef.

1	Start	→	OptFunctions PrimDescs | ϵ
2	OptFunctions	→	**functions** {% *raw text* %} | ϵ
3	PrimDescs	→	PrimDesc PrimDescs | PrimDesc
4	PrimDesc	→	**primitive** *identifier* **: size** (*float* , *float*)
5			OptOrigin OptNorm { PrimDescBlocks }
6	OptOrigin	→	**normalise** (*float*) | ϵ
7	OptNorm	→	**origin** (*float* , *float*) | ϵ

Where PrimDescBlocks *is the grammar symbol defined in Figure 6.13.*

Figure 6.15: The PRIMELA syntax to describe a complete primitive.

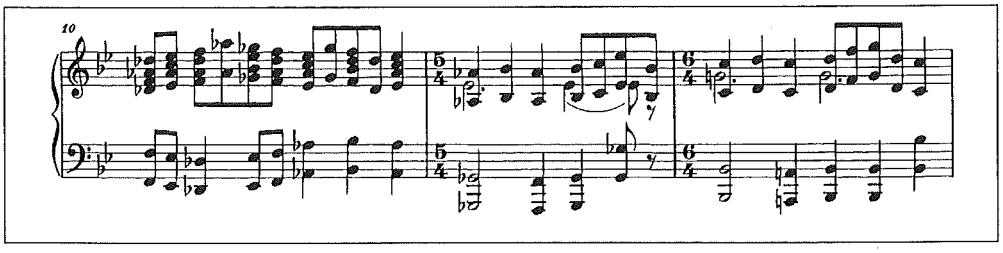


Figure 6.16: An example excerpt of music written using CMN.

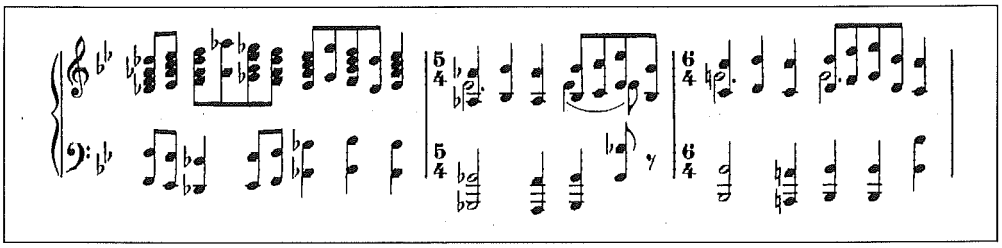


Figure 6.17: The starting point for the primitive detection stage of the OMR system.

6.8 Evaluation

The aim of the primitive detection stage for our OMR system is to process arbitrary graphical shapes from an arbitrary music notation, whilst minimising the development time.

To illustrate the success of PRIMELA, two sample works were selected from the corpus: one using CMN and the other using plainsong notation. The results presented below show the primitive shapes detected, and the time taken to check the image for each primitive. The section ends with a comparison of the code size required to recognise the same set of primitives using a traditional programming language. Tabulation of the recognition rates from processing a larger collection of works is deferred until Chapter 9, where we study the entire OMR process.

6.8.1 Common music notation

Figure 6.16 shows an excerpt of CMN from the corpus entry "Promenade." Processing by the staff detection and the object location stages, as well as correcting for skew, leads to the image shown in Figure 6.17. This is the starting point for primitive detection.

For this example, primitive descriptions that detect treble clefs, flats, naturals, long

Primitive type	Description
Treble clef	Horizontal projection applied to objects. No extensions.
Flat	Template match applied to objects. Unknown object extended to fit template, plus a 10% extension in both axes. Post condition checks for one black section in a horizontal slice near the top of the shape, and two black sections in a horizontal slice near the bottom.
Natural	Template match applied to objects. Unknown object extended to fit template, plus a 10% extension in both axes. Post condition checks for one black section in a horizontal slice near the top of the shape, and one black section in a horizontal slice near the bottom.
Long vertical line	Hough transform applied to objects. Only applied when starting pixel under consideration is black. No extensions.
Short vertical line	Hough transform applied to objects. Only applied when starting pixel under consideration is black. No extensions.
Bass clef curl	Hough transform applied to objects. Only applied when starting pixel under consideration is black. Unknown object extended to fit parametric equation, plus a 50% extension to the right and down. Match is performed using the original bitmap.
Beam	Connectivity analysis applied to objects. Post condition checks the object is thicker than a slur with negligible curvature.
Filled-in (black) note head	Template match applied to objects. Only applied when starting pixel under consideration is black.

Table 6.5: Summary description of the detection methods used for each CMN primitive type.

vertical lines, short vertical lines, bass clef curls, beams, and filled-in (black) note heads were used. Images showing the shapes located for the respective primitive types are shown in Figures 6.18 to 6.25. There is no musical distinction between short vertical lines and long vertical lines. Depending on its height, a note stem can be classified as either a short vertical line, or a long vertical line. The same is true for a bar line, and so on. The decomposition is made to simplify the task of describing all possible vertical lines (see Section 9.2.2 for a fuller explanation). Table 6.5 summarises the detection methods used for each primitive type.

For this image, recognition rates are high, with only four mistakes being made. Since there was no description for a curly brace ({) the OMR system mistakenly identifies two long vertical lines in the curly brace to the left of the staff system. Also, the certainty rating for one of the naturals was only 85% (it is shown as gray in Figure 6.20),

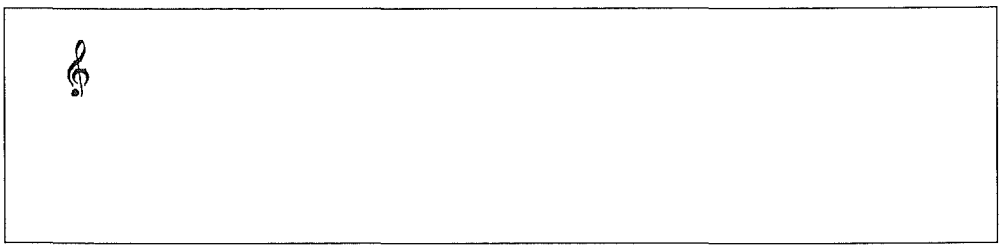


Figure 6.18: The objects detected as treble clefs.

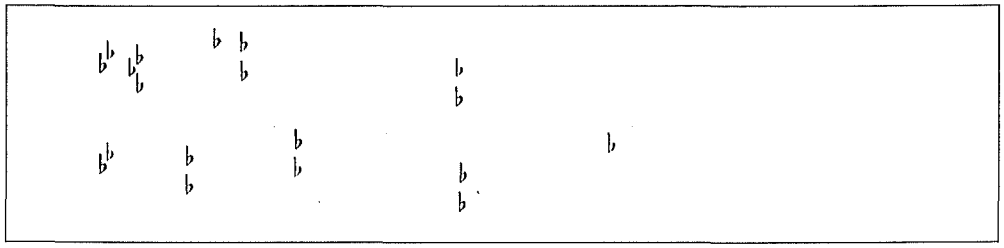


Figure 6.19: The objects detected as flats.

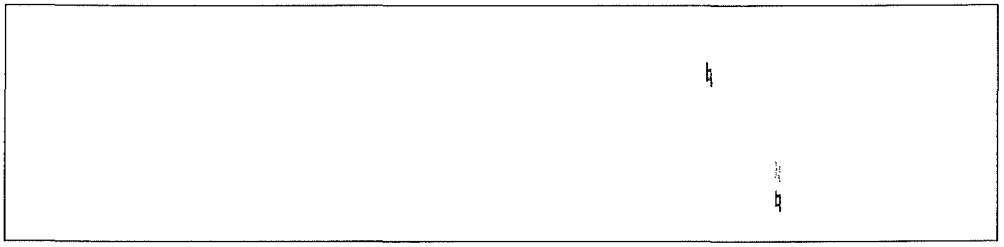


Figure 6.20: The objects detected as naturals.

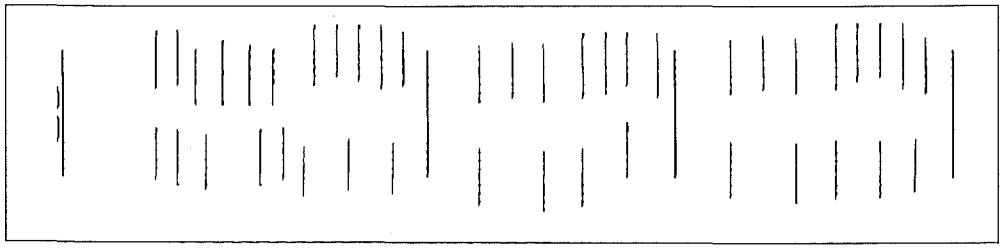


Figure 6.21: The objects detected as long vertical lines.

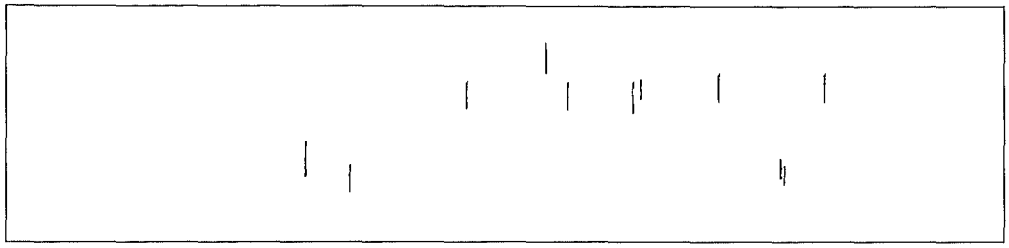


Figure 6.22: The objects detected as short vertical lines.

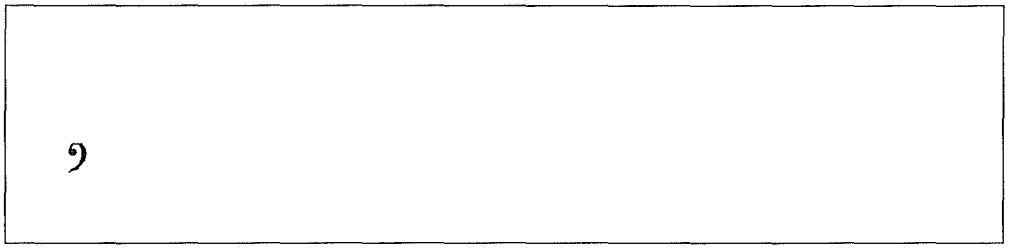


Figure 6.23: The objects detected as bass clef curls.

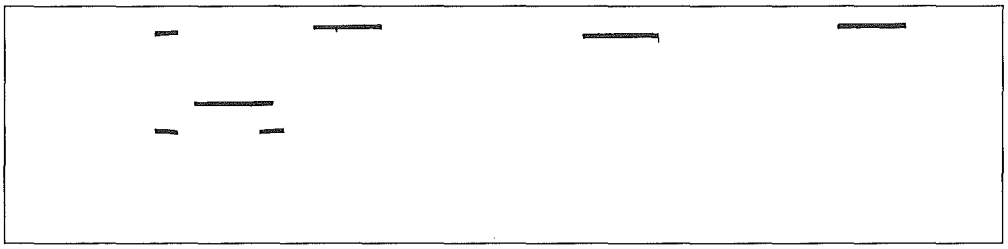


Figure 6.24: The objects detected as beams.

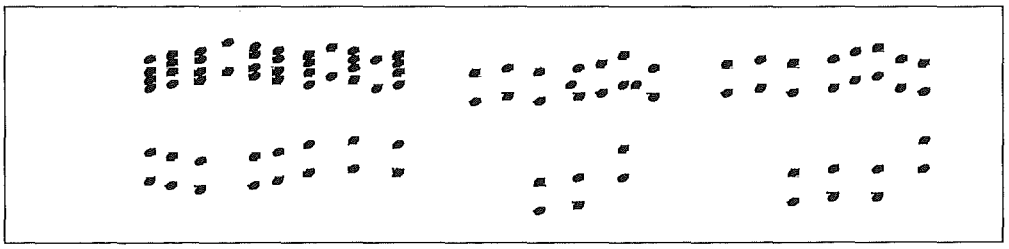


Figure 6.25: The objects detected as filled-in (black) note heads.

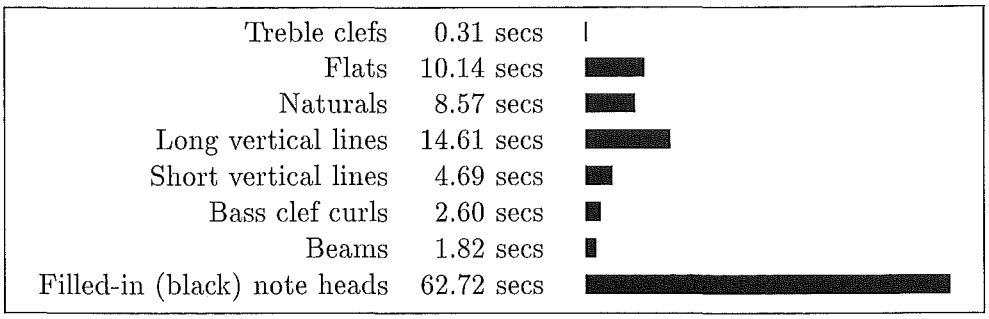


Figure 6.26: Timing information for processing primitives in the CMN example.

and consequently the shape was not removed, allowing two short vertical lines to be detected as possible alternatives. Neither of these types of mistake cause errors during the assembly process. In the case of the curly brace mistake, the erroneous vertical lines fall outside the horizontal scope of the staff system, and are therefore discounted by the assembly process as neither potential bar lines nor note stems. For the two erroneous short vertical lines found within the bottom natural, their lengths are too short for the assembly process to consider them as bar lines or note stems.

Processing times for detecting these primitives are given in Figure 6.26. Clearly the detection of filled-in note heads is most expensive, taking longer than all the other processing times added together. There are two reasons for this. First the number of positions checked is extremely high, and second, template matching is a computationally expensive operation. Optimisations for template matching exist. In the OMR system by Reed [Ree95], the cost of note head detection is reduced by designing a set of templates that sample the original template at a lower resolution. The standard one-pass algorithm is replaced with a multiple pass algorithm, and successive templates are tested. Should a lower resolution template be encountered that records "no match," then the algorithm terminates immediately. Making this alteration, Reed observed a speed up of 67% for the detection of filled-in note heads.

6.8.2 An example image in plainsong notation

To evaluate the flexibility of the system, a set of PRIMELA descriptions was written for plainsong notation. Figure 6.27 shows an excerpt of plainsong notation from the corpus entry "Extra Tempus Paschale." After processing by the preceding OMR stages, Fig-

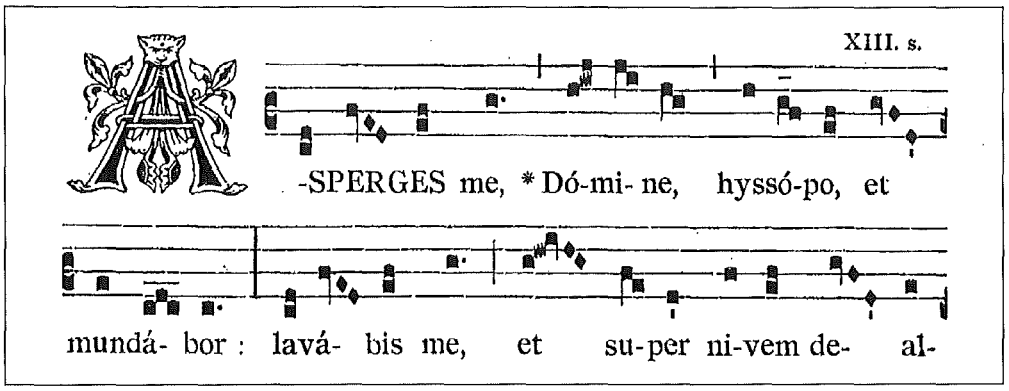


Figure 6.27: An example excerpt of music written using plainsong notation.

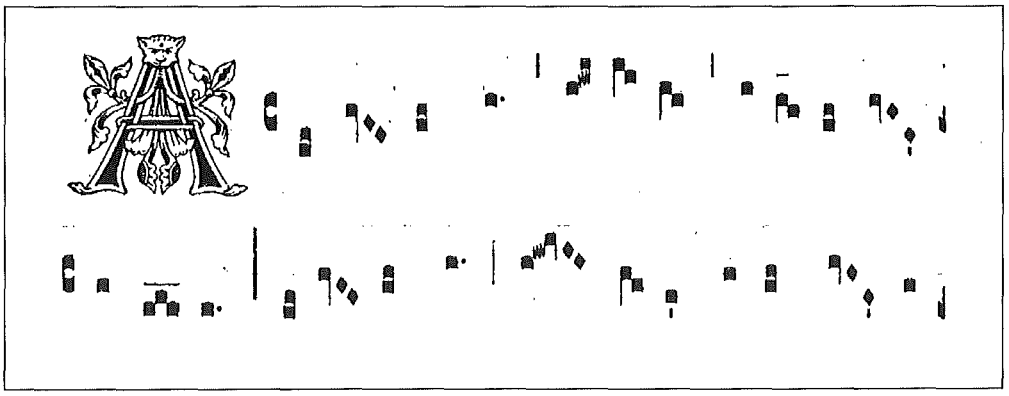


Figure 6.28: The starting point for the primitive detection stage of the OMR system.

ure 6.28 is the image presented for primitive detection.

For this example, primitive descriptions that detect long vertical lines, horizontal lines, rectangles, rhombuses, short vertical lines, and dots were used. Images showing the shapes located for the respective primitive types are shown in Figures 6.29 to 6.34. Like the vertical line descriptions for CMN, there is no musical distinction between short vertical lines and long vertical lines, and the decomposition only exists to simplify the task. Table 6.6 summarises the detection methods used for each primitive type.

For plainsong notation the sample image is of good quality, and recognition rates are high. Some minor mistakes are made in identifying rectangles, rhombuses, and dots. One of the rectangles on the first staff is matched with an 80% certainty rating because it is smaller in size than the others, and on the second staff a saw-toothed shape is tagged as a potential rectangle with a 60% certainty rating (they are shown as gray in Figure 6.31).

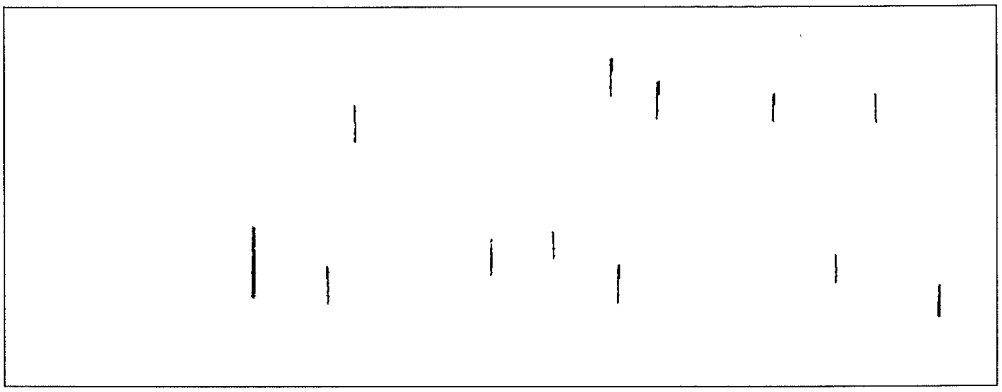


Figure 6.29: The objects detected as long vertical lines.

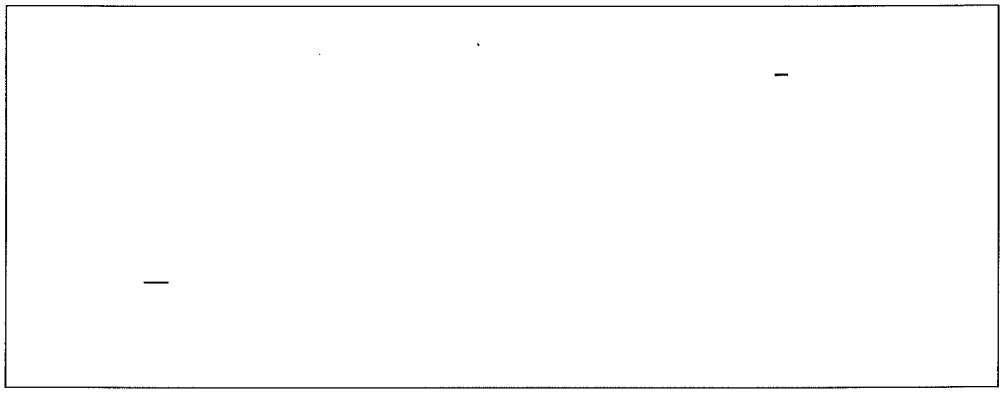


Figure 6.30: The objects detected as horizontal lines.

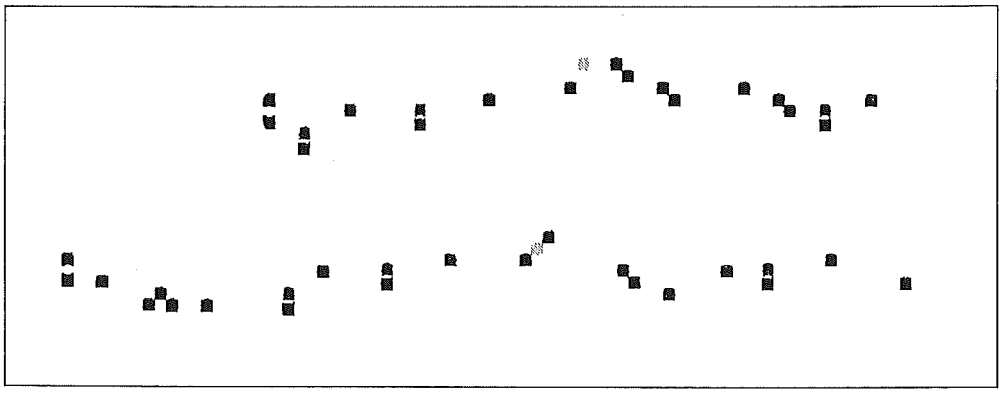


Figure 6.31: The objects detected as rectangles.

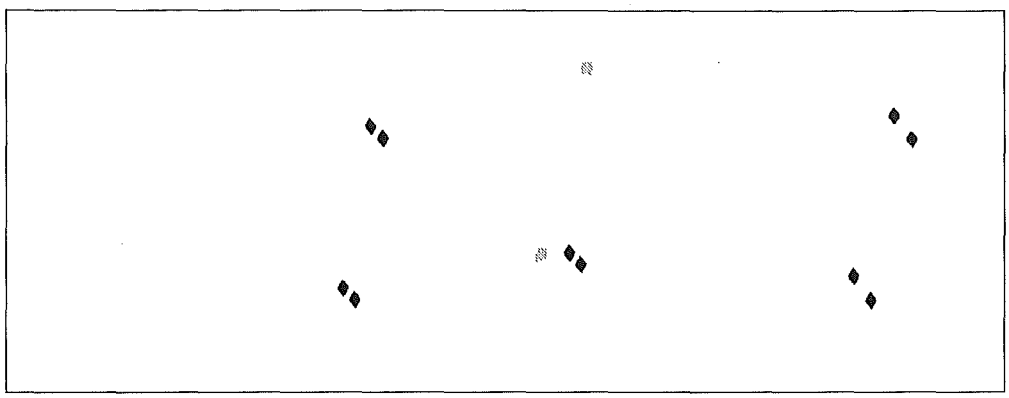


Figure 6.32: The objects detected as rhombuses.

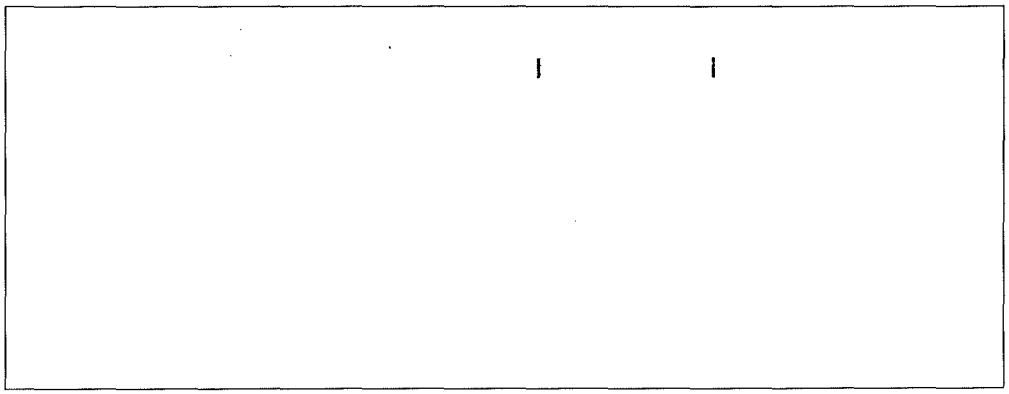


Figure 6.33: The objects detected as short vertical lines.

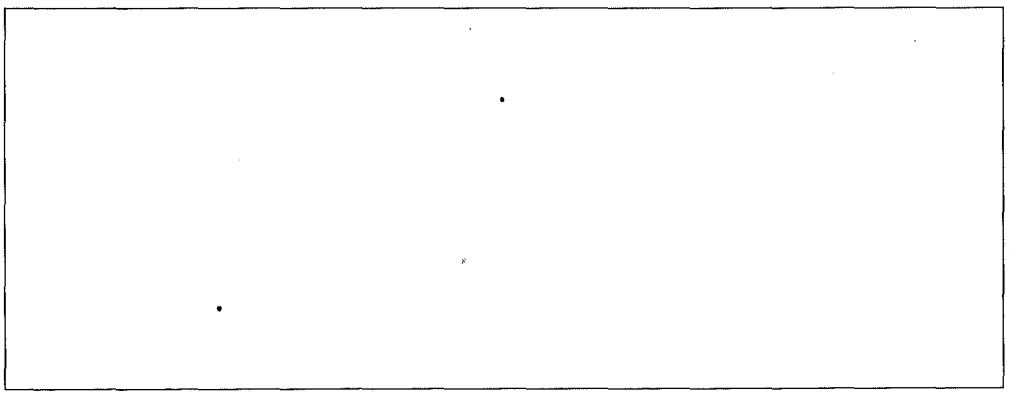


Figure 6.34: The objects detected as dots.

Primitive type	Description
Long vertical line	Hough transform applied to objects. Only applied when starting pixel under consideration is black. Unknown object extended vertically to fit parametric equation.
Horizontal line	Hough transform applied to objects. Only applied when starting pixel under consideration is black. Unknown object extended horizontally to fit parametric equation.
Rectangle	Template match applied to objects. Only applied when pixel under consideration is black. Unknown object enlarged by 10% in both axes.
Rhombus	Template match applied to objects. Only applied when pixel under consideration is black. Unknown object enlarged by 10% in both axes.
Short vertical line	Hough transform applied to objects. Only applied when starting pixel under consideration is black. Unknown object extended vertically to fit parametric equation.
Dot	Template match applied to objects. No extensions.

Table 6.6: Summary description of the detection methods used for each plainsong notation primitive type.

The same two shapes are tagged as potential rhombuses with certainty ratings of 10% and 20% respectively. When detecting dots, one is slightly smaller than the others, resulting in a certainty rating of 50%. Such errors are to be expected when processing plainsong notation because printers did not have the same printing equipment that is available today. Certainty ratings add robustness to the OMR system, allowing the primitive assembly stage to correctly resolve such recognition conflicts.

A rogue long vertical line is included in Figure 6.29 at the end of the second staff. Figure 6.35 shows an enlargement of this area taken from the original image. In plainsong notation it is common practice to partially draw the first note of one staff at the end of the previous staff, as a cue for the singer. This occurs in the sample image, and for the last shape on the second staff, the right-hand side of the shape is matched as a dubious vertical line, but the half drawn note head is not recognised at all. For the cueing note at the end of the first staff, no primitives are detected. Providing PRIMELA descriptions for cue notes avoids such mistakes.

Processing times for detecting the primitives are given in Figure 6.36. Template matching for rectangle and rhombus note heads is substantially faster than the detection of filled-in note heads in CMN because objects containing note heads in this notation do not have such a diverse range of sizes as they do in CMN, and consequently a tighter

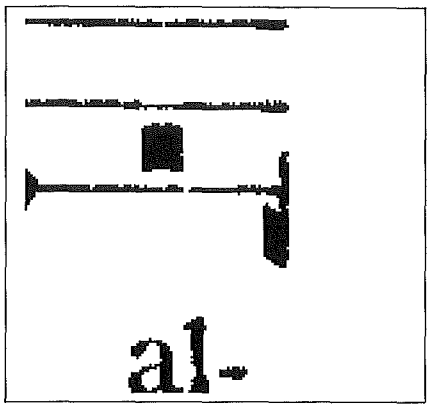


Figure 6.35: An enlargement of the end of the second staff, taken from the plainsong notation example.

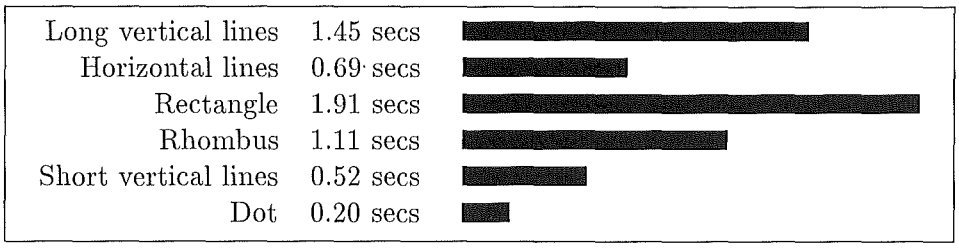


Figure 6.36: Timing information for processing primitives in the plainsong notation example.

bounding box check can be made as a pre-condition to template matching. Despite having a slightly larger template, detecting rhombuses is faster than rectangles. This is because the rectangles have already been removed by the time the check for rhombuses is made, and therefore there are fewer black pixels in the image that meet the "pixel under consideration must be black" requirement (Table 6.6).

6.8.3 Using a traditional programming language

To compare the length of PRIMELA descriptions with equivalent versions written in a traditional programming language, two dedicated programs (one for CMN and the other for plainsong notation) were implemented using C++ to recognise the same set of primitives processed in the above examples. Table 6.7 tabulates the average length of the code per description. The values are calculated from the code written purely for individual primitive types, discounting overheads such as the pool of pattern recognition techniques and

Type of notation	C++	PRIMELA
Common music notation	928 lines	56 lines
Plainsong notation	1036 lines	52 lines

Table 6.7: Average number of lines of code required per description to detect the primitives from the above examples.

the code for the graphical user interface, which are constant for both the PRIMELA system and the dedicated programs.

Using PRIMELA the reduction in code size is dramatic. In addition to the time saved writing the code, less code reduces the chance of writing an incorrect description (a bug!), and even if a mistake is made, the short length of the code makes it easier to track down.

Chapter 7

Primitive assembly

It was stated earlier that the classification of musical features is simplified by decomposing the problem into the detection of the primitives shapes, followed by the assembly of these shapes into their respective musical features. In the previous chapter we discussed primitive detection. In this chapter we discuss primitive assembly.

The essential task is that of expressing a valid taxonomy of musical features. By this we mean how the detected primitive shapes, such as filled-in note heads, vertical lines, and sharps are assembled into musical features such as notes, and key signatures. In most existing OMR systems this is accomplished with hard-wired code, but this is insufficient for an extensible system. What is required is a formal knowledge representation that the assembly process can access.

In this chapter we first review three knowledge representation schemes that are well suited to describe structured objects: production rules, semantic networks, and frames. After considering the strengths and weaknesses of these approaches, no individual scheme stands out as best for the task at hand. The production rule scheme was chosen due to the author's familiarity with its syntax.

The main body of this chapter describes the assembly system devised. Using a variant of the Definite Clause Grammar (DCG) technique, rules encode a valid taxonomy and the matching order of musical features. The collection of primitives detected in the image—which represent the tokens of the language—are parsed according to the grammar, resulting in a series of annotated derivation trees, where each tree represents one musical feature.

Results are presented at the end of the chapter. To demonstrate the flexibility of the design, two grammars were implemented: one for CMN, and the other for plainsong notation.

7.1 Knowledge representation schemes

The topic of knowledge representation in computer science has been, on the whole, application driven. Much of the literature on this topic takes the form of individual papers that describe the particulars of a knowledge representation scheme for a specific application. Fortunately, overviews also exist that generalise the trends in existing work, for example "The Handbook of Artificial Intelligence" [BF81] and "Principles of Artificial Intelligence" [Nil82].

In the following sections we discuss three formal methods for knowledge representation that are well suited to describing the structured taxonomy of musical features: production rules, semantic networks, and frames. Unfortunately, due to the nature of the topic, there is no standard notation used to explain these schemes; nor, for that matter, are there rigid boundaries to the mechanisms included in a scheme. In the discussion below the *essence* of each scheme is described.

7.2 Production rules

Production rules are a popular scheme for representing knowledge [DK77, BF81, Bra90, Li91]. A common implementation is to use **if-then** rules:

if Precondition $P1$ **and** Precondition $P2$ ... **then** Conclusion C.

A single rule encodes a small, typically independent, piece of knowledge. A collection of rules form a knowledge-base. Both preconditions and conclusions evaluate to boolean values, so a conclusion from one rule can be used as a precondition for another, and so on.

To simplify accessing data in the knowledge-base, most systems restrict the syntax of the **if** statement so that it can only contain a conjunction (boolean AND) of preconditions, where a precondition may use negation (boolean NOT). Since a collection of **if** statements form a disjunctive (boolean OR) sequence, a knowledge-base takes a "sum of

```
if pitch(Sharp)='F' and adj(Treble_clef,Sharp)   then G_sig
if pitch(Sharp)='F' and adj(Bass_clef,Sharp)     then G_sig
if pitch(Sharp)='F' and adj(Bar_line,Sharp)      then G_sig
if G_sig   and pitch(Sharp)='C'  and adj(G_sig,Sharp)   then D_sig
if D_sig   and pitch(Sharp)='G'  and adj(D_sig,Sharp)   then A_sig
if A_sig   and pitch(Sharp)='D'  and adj(A_sig,Sharp)   then E_sig
if E_sig   and pitch(Sharp)='A'  and adj(E_sig,Sharp)   then B_sig
if B_sig   and pitch(Sharp)='E'  and adj(B_sig,Sharp)   then F#_sig
if F#_sig  and pitch(Sharp)='B'  and adj(F#_sig,Sharp)  then C#_sig
```

Figure 7.1: A knowledge-base using **if-then** rules that represents how key signatures involving sharps are assembled.

products" structure, which due to De Morgan's theorem is equivalent in expressive power to an unrestricted syntax.

Given a starting set of conditions, the rules of the knowledge base can be used to deduce what conclusions are valid. Alternatively, a conclusion can be singled out and checked to see if it fits the starting conditions. Such inference mechanisms are known as *forward* and *backward* chaining, respectively.

As an example, consider expressing the knowledge necessary to assemble accidentals into key signatures. For the sake of brevity we will only consider the case of sharps. Figure 7.1 shows one possible solution using the **if-then** construct, and relies on two utility functions: `pitch` and `adj`. The function `pitch` takes the given shape, and returns its pitch as a character; the function `adj` takes two shapes and determines if they are adjacent or not.

Taking the detected primitives as our starting point, we can deduce what key signature is present at a particular place in the image using the knowledge-base. In the case of forward chaining, more than one conclusion can be drawn from the data. For example, from a piece of music in the key of A, the knowledge-base can infer three key signatures: G, D, and A. In some situations drawing more than one conclusion is desirable, but not in the case of key signatures.

This example raises the important issue of *controlling* access to the knowledge-base. For most applications, traversing the knowledge-base to produce every possible conclusion is not what is required, and in most practical situations this would be intractable anyway. An application, therefore, controls access to the knowledge-base. In the case of the key-

signature example, the application program should be restricted to finding the longest inference path through the knowledge-base.

As it stands the example knowledge-base could not be used in the OMR framework proposed for this thesis. To calculate the *pitch* of the sharps, the OMR system must have processed the scope of the detected clefs; but this is a musical semantics step which, in our framework, is defined to occur after primitive assembly. This is not a critical point since it is possible to replace the reliance on pitch with the positional height of a sharp on the staff.

Grammars

Production rules can also be expressed as grammars. A production rule in a grammar is of the form:

$$\mathrm{C} \quad \rightarrow \quad P_1 P_2 \ldots .$$

From the point of view of a knowledge-base, this is merely an **if-then** rule conforming to a different syntax. The symbol on the left-hand side, C, is dependent on the symbols on the right-hand side, P_1, P_2, ... In other words, **if** the preconditions on the right-hand side are met, **then** the conclusion can be asserted.

In Figure 7.2 we recast the key signature example from above as a grammar. Since the example is mostly a syntactic reshaping, it is not surprising to see the utility functions `pitch` and `adj` reused. The example also relies on the common extension in grammars of embedding code. Such sections in the example are enclosed in curly braces, { ... }, and are assumed to be boolean in evaluation.

7.3 Semantic networks

A semantic network is a knowledge representation scheme based on a directed graph [BF81, Nil82, Tan87, Bra90, Sow91]. In the basic network, nodes represent entities, and the edges between nodes represent set membership. An example of this is shown in Figure 7.3, where the set membership of some musical features are expressed as a semantic network. As the figure illustrates, a semantic network naturally expresses *inheritance*—a fundamental structure in many knowledge domains.

On its own, an inheritance structure is limited in the knowledge representation problems it can express. This is why semantic networks are augmented with constructs that

```
Key_sig  →  G_sig | D_sig | A_sig | E_sig | B_sig | F#_sig | C#_sig
  G_sig  →  sharp { pitch(sharp)='F' } { adj(Treble_clef,sharp) }
  G_sig  →  sharp { pitch(sharp)='F' } { adj(Bass_clef,sharp) }
  G_sig  →  sharp { pitch(sharp)='F' } { adj(Bar_line,sharp) }
  D_sig  →  G_sig sharp { pitch(sharp)='C' } { adj(G_sig,sharp) }
  A_sig  →  D_sig sharp { pitch(sharp)='G' } { adj(D_sig,sharp) }
  E_sig  →  A_sig sharp { pitch(sharp)='D' } { adj(A_sig,sharp) }
  B_sig  →  E_sig sharp { pitch(sharp)='A' } { adj(E_sig,sharp) }
 F#_sig  →  B_sig sharp { pitch(sharp)='E' } { adj(B_sig,sharp) }
 C#_sig  →  F#_sig sharp { pitch(sharp)='B' } { adj(F#_sig,sharp) }
```

Figure 7.2: A knowledge-base using grammar rules that represents how key signatures involving sharps are assembled.

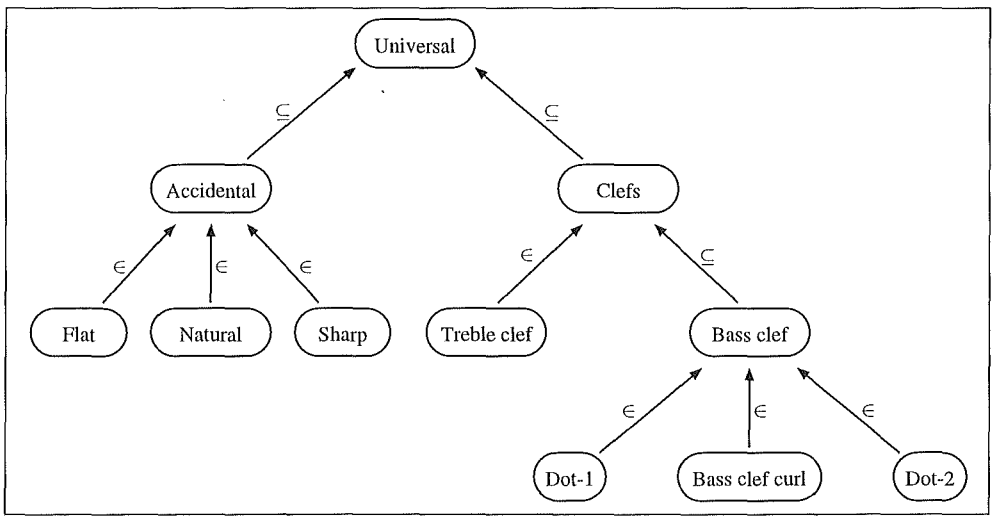


Figure 7.3: An example of a basic semantic network.

express a *function* of an entity, and a *relationship* between entities.

An instance of a *function* is included in a semantic network by simply annotating the network with the desired values: a labelled arc representing the function name is added from the domain (the entity) to the range (the desired value). Typically a function is used to represent a property of an entity. In Figure 7.4 the previous example has been annotated with the property *vertical position*. To aid clarity in this figure, the descendents from the Bass clef node have been omitted.

Musical features such as sharps, flats and naturals can appear at various vertical positions on a staff. This property is represented in Figure 7.4 with new arcs connecting

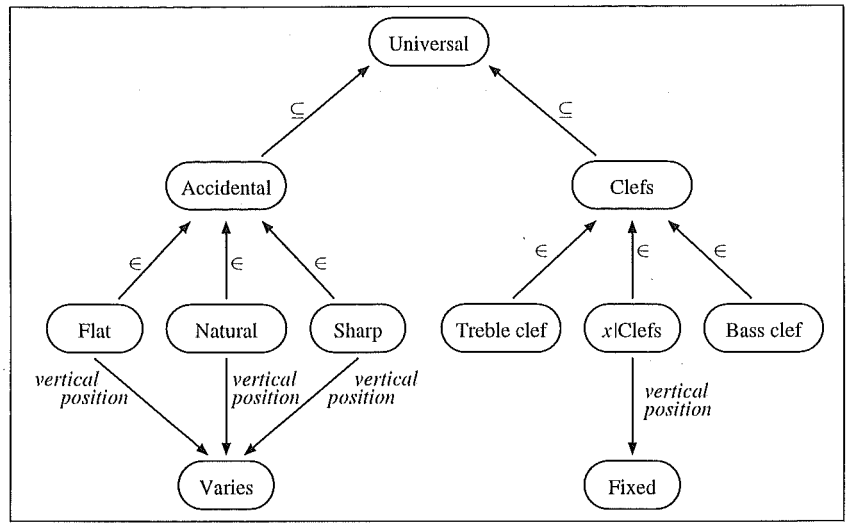


Figure 7.4: An example semantic network annotated with the *vertical position* property.

these musical features to the property node, Varies. Musical features such as a treble clef and a bass clef are fixed in the position they can appear vertically on a staff, so we could represent this knowledge in the same manner as before. However, if a property applies to a set of entities, then a *delineation* node can be used to annotate the set, instead of individually annotating each entity. This has been done in the case of the Clefs node, where a delineation node (x|Clefs) has been added, connecting the Clefs node to the property node, Fixed. In fact we would normally represent the vertically free property of accidentals in the same way. The property was represented as individual arcs to illustrate both constructs in the example.

Expressing a *relationship* in a semantic network is more convoluted than a function, since it requires the network to be extended by the concept of a *case frame*. In a case frame, the complete relationship is represented as a single node; elements of this node are instances (cases) of the relationship, and are nodes themselves; descendents from case nodes detail the particular entities involved in an instance of the relation, using labelled arcs in an identical manner to an instance of a function. In Figure 7.5 the relationship *adjacent* is expressed as a semantic network: nodes Adj-1 and Adj-2 are case nodes, and nodes Dot-1, Dot-2 and Bass clef curl detail the particular entities involved in each case.

Using these constructs, let us now revisit the key signature example. Figure 7.6 shows a semantic network that is one possible representation of the knowledge. The pro-

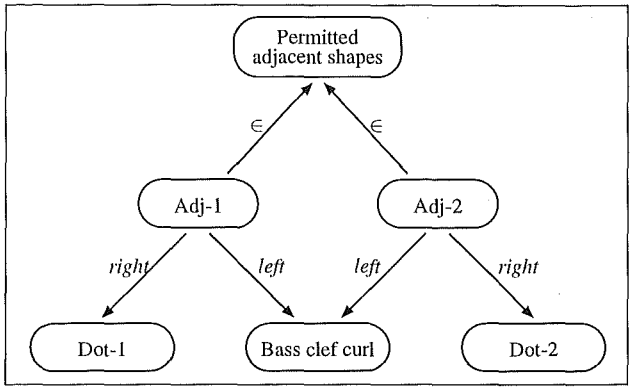


Figure 7.5: An excerpt from a semantic network that shows the *adjacency* relationship.

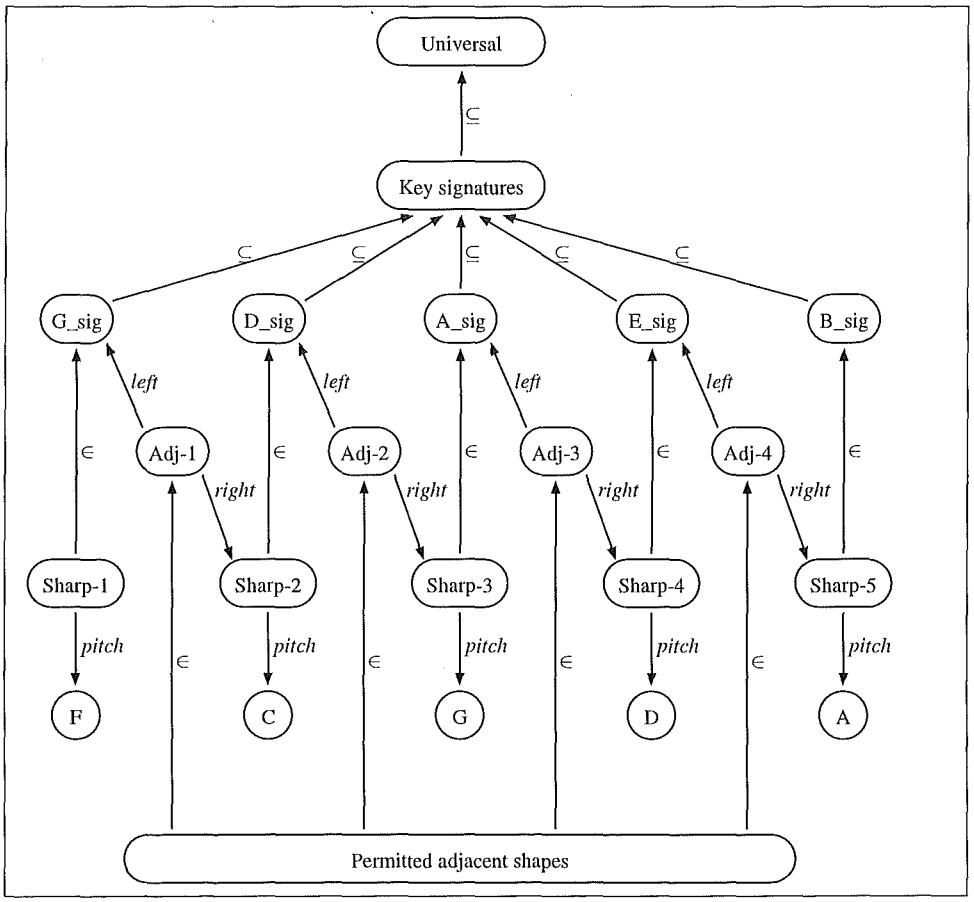


Figure 7.6: A semantic network that represents how key signatures involving sharps are assembled.

duction rule representation for key signatures relied on two utility functions `pitch` and `adj`. In the semantic network these utilities appear as a function in the former case (*pitch*), and as a relation in the latter case (*Permitted adjacent shapes*). The last two key signatures, F# and C# have been omitted from the figure to keep it simple.

Like production rules, a semantic network on its own does not solve any problems. Problems are solved by developing applications that access the knowledge-base. In the case of a semantic network, this is normally done by taking known facts from the problem domain and trying to match them against parts of the semantic network. In terms of assembling the primitive shapes of music notation, such a strategy becomes the iterative matching of the primitive shapes detected in the image with the network, to see what larger shapes can be deduced.

7.4 Frames

Frames take an object oriented approach to knowledge representation. In keeping with this paradigm, inheritance is represented *explicitly* and the access control to the knowledge is dispersed throughout the objects. This contrasts with semantic networks where access control to the knowledge structure is external, and consequently inheritance is *implicit* in how the access control chooses to interpret the links between nodes. Like other knowledge-based schemes, there is no standard form for frames, although common concepts recur [Min85, BF81, Nil82, Tan87, Bra90].

As the name suggests, the main structure used in this scheme is a *frame*. A frame is a collection of knowledge relevant to a particular entity, situation, or concept, where an individual item in a frame is represented using a *slot* which can be a basic data-type or a reference to another frame. A slot itself consists of two attributes: the *slot name*, and the *value* to be filled in.

A simple example is given in Figure 7.7 which shows two frames relating to the dynamic volume of a piece of music. The first frame, *Volume*, forms the basis for the concept of volume; the second frame, *Fortissimo*, details the particular instance of a volume setting, and is inherited from the base frame. In the second frame, the value for the slot Volume is set explicitly, whereas the inheritance mechanism is relied upon to represent the knowledge that the fortissimo setting affects notes. In this simple example, the details for

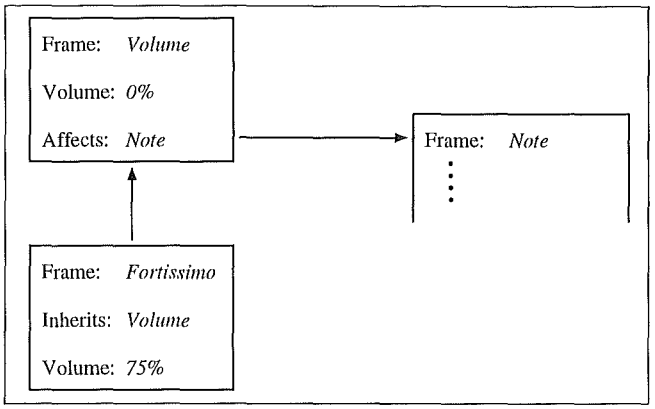


Figure 7.7: A simple example of knowledge representation using frames.

the frame, *Note*, have been omitted. In a larger, complete system, the note frame would lead to a complex structure of frames where different types of notes—crotchet, minim, semibreve and so on—are inherited from the basic note frame.

Further sophistication is added to a frame through *attachments*. Three important types of attachment are: *constraint*, *if-needed*, and *if-added*. A *constraint* attachment must be satisfied before a slot is filled. An *if-needed* attachment specifies a procedure that computes the value of the slot, if needed. An *if-added* attachment specifies a procedure to execute if a slot has just been filled.

A frame-based solution to the key signature example is shown in Figure 7.8. Once again we see the same utility functions are relied upon to compute pitch, and to confirm that two shapes are adjacent. In this situation the utility functions form constraint requirements (Con:) attached to the slot, Last_acc. Only the first four key signatures are represented in the figure.

7.5 Choosing a knowledge representation scheme

There are many factors that influence the choice of a knowledge representation scheme [CM87]. The main considerations are: *knowledge elicitation*, *expressive adequacy*, and *notational efficacy*.

In most knowledge representation problems, knowledge elicitation is a difficult process. The implementor of the knowledge-base usually has computer expertise, but little or no knowledge of the problem domain, and therefore must rely on literature and spe-

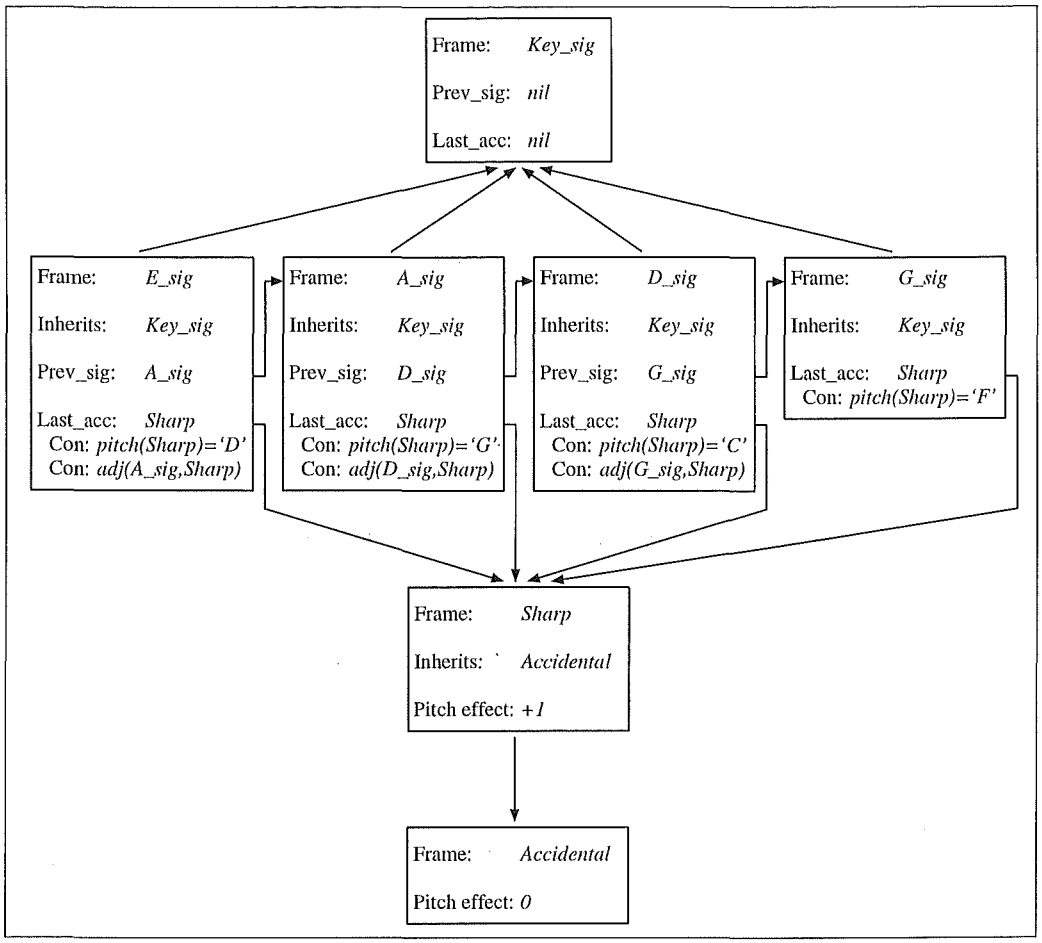


Figure 7.8: A frame-based scheme that represents how key signatures involving sharps are assembled.

cialists in the field, with the complications in communication this entails. In our situation, the issue is simplified somewhat by the author being both the implementor of the knowledge-base and the expert in music notation. The main considerations, therefore, are expressive adequacy and notational efficacy.

Most musical features are *geometrically constrained* in where the primitive components to the shape can occur. For example, a duration dot that accompanies a "basic note"[1] must be situated immediately to the right of the note, at a vertical position that is centred on the note head that the durational dot affects. All three knowledge representation schemes under consideration have variants that include constraints, consequently all are capable of representing this attribute of music notation.

[1] A filled-in black note head and stem.

However, not only are most musical features geometrically constrained, but these geometrically constrained structures are *common* to different musical features. For example, a quaver note includes the same structure as a "basic note" with the addition of a tail on the right-hand side of the stem; similarly for a semiquaver note, and so on. An abstract concept that captures this attribute of music notation is *inheritance.* In the case of semantic networks and frames, inheritance is fundamental to both schemes. In the case of production rules, inheritance is a matter of viewing the rules in a particular way. Taken from the **if-then** example for key signatures (Figure 7.1) the rule,

if E_sig **and** `pitch(Sharp)='A'` **and** `adj(E_sig,Sharp)` **then** B_sig

for the key signature B, can be seen as inheriting all the assembly requirements for a key signature in E, plus a new sharp at pitch A that is adjacent to it. Consequently, all three schemes are capable of representing common structures in music notation through inheritance.

Finally, in addition to geometrically constrained structures being common in different musical features, a geometrically constrained structure can be *repeated an arbitrary number of times* within a musical feature. For example, a beamed note includes repeated instances of the structure previously described as a "basic note." To represent this attribute of music notation, a knowledge representation scheme must be able to express *iteration.* The purest form of this is *recursion*; other programming mechanisms that represent this are looping constructs and lists. The three representation schemes under consideration all have extensions that permit the representation of iteration, and consequently repeated sub-structures in music notation.

This means that all of the representation schemes under consideration can adequately describe the taxonomy of music notation. All that remains is to consider notational efficacy.

The taxonomy of musical features is strongly structured, as are the three representation schemes under consideration. If an expert in music notation were to explain musical feature taxonomy to a novice, one can see how diagrams resembling the structures that our three schemes encapsulate could be drawn to aid the explanation. All three schemes, therefore, closely resemble the abstract knowledge we understand as the taxonomy of musical features.

So, having considered the main influencing factors in choosing a knowledge representation scheme, there is no distinctive winner for expressing the taxonomy of musical features. The choice, therefore, comes down to the syntax that the designer is most comfortable with. For this work, the grammar form of production rules was chosen.

7.6 Existing grammar based OMR systems

It was stated earlier that most OMR systems implement primitive assembly and musical semantics with hard-wired code, and are therefore static in the set of music notation they can process. There are, however, three notable exceptions where these stages of the system are dynamic in their capabilities. All are based on grammars.

7.6.1 Graph grammars

An attributed programmed graph grammar [Bun82] forms the basis for the work by Fahmy and Blostein [FB91, FB92]. In this system a grammar is used to simultaneously assemble musical features from primitive shapes and to compute semantic information, such as note pitch and duration.

There is a hierarchy of grammar methodologies, based on the type of data-structure they generate. Commonly used data-structures are strings, arrays, trees, and graphs—increasing in expressive power as listed, but also increasing in the complexity of the parser required. String grammars form the basic model and are frequently used to specify computer languages for compilers; however, they are not particularly suited to two-dimensional problems. The use of data-structures that are themselves two-dimensional, model structured document problems more naturally [San92]. The most powerful structure is a graph grammar [EKR90]. Like string grammars, where restrictions in the type of grammar—context sensitive, context free, and so on—permit more efficient parsers, graph grammars are often restricted to sub-classes, since this reduces the degree of complexity in parsing such languages.

Traditionally a graph grammar translates an input graph into an output graph that reflects a higher level of understanding of the document. The work by Fahmy and Blostein differs slightly, in that the starting point is a collection of isolated nodes that represent the primitive shapes detected in the image. The graph grammar processes these nodes, form-

Backus-Naur Form	Definite Clause Grammar
S → a b	`s --> [a], [b]`
S → a S b	`s --> [a], s, [b]`

Table 7.1: A Definite Clause Grammar is a Backus Naur Form grammar with different syntax.

ing links that represent interactions between primitives, generating a connected graph as output. Storing attributes at nodes is also under the control of the graph grammar; in this way semantic information can be calculated simultaneously.

Initially Fahmy and Blostein's system used a "Build-Weed-Incorporate" model to resolve conflicts that can arise when forming links. Later this was altered to a "Build-Constrain-Incorporate" model when the system was extended to cope with the more realistic scenario of uncertain data. Uncertainty was introduced into the system by tagging primitives with values reflecting how reliable the match was.

In the system developed by Reed, the work by Fahmy and Blostein is used as a foundation for an alternative algorithm [Ree95]. The graph grammar methodology that transforms the input graph of isolated nodes into a connected graph remains, however, Reed merges this stage of the OMR process with the primitive detection stage. Consequently, contextual information can be used to guide the search for primitives.

7.6.2 Definite clause grammars

An extended version of Definite Clause Grammars forms the basis of the work by Coüasnon *et al.* [CC94, CBS95]. In this system, a grammar forms the top level control to the OMR system. The primary role of the grammar is to specify the taxonomy of musical features, but it also controls the joining and segmentation of shapes, as well as defining a sequential order for processing objects on a staff—effectively specifying how to "read" music.

Put simply, a Definite Clause Grammar (DCG) is a Backus Naur Form (BNF) grammar conforming to slightly different notational conventions [Bra90, ASU86]. In Table 7.1 a BNF grammar is compared with an equivalent DCG. The motivation for this change in syntax comes from Prolog programming, where an automatic process can translate a DCG into Prolog clauses. Combining a DCG with the facilities offered by Prolog leads to a powerful platform, permitting the elegant implementation of parsers.

1	NoteGr	→	NoteGrU | NoteGrD
2	NoteGrU	→	**stem** ***closel_downtip*** NoteHead
3	NoteHead	→	Head ***closel_same_line*** [Accidental]?
4	Head	→	**black_note_head** | **white_note_head**
5	Accidental	→	**natural** | **flat** | **sharp** | **double_flat** | **double_sharp**

Figure 7.9: A simplified excerpt of the grammar used by Coüasnon *et al.*

Coüasnon *et al.* augment this notation with two additional constructs to aid their task of assembly, segmentation, and sequential ordering: a *position operator* takes the form $A\ \boldsymbol{P}\ B$ and means A and, at position $\boldsymbol{P}$ in relation to A, we find B; and *factorisation* which takes the form $A\ (\ B\ :\ C\ )$ and means $A\ B$ and $A\ C$. The implemented translator converts a specified grammar into λProlog—a higher order dialect of Prolog—as this language more naturally supports the features of the extended DCG.

Based on an example presented by Coüasnon *et al.* [CBS95], Figure 7.9 shows the grammar rules for a crotchet or minim note that can optionally include an accidental in front of the note head. The example is only intended to illustrate the principle, and thus has been simplified by omitting the productions for chords, dots, accents, and slurs. BNF syntax is used to express the grammar for convenience, since it is trivial to change this to the extended DCG. The production rule on line 2 can be interpreted as, "There is a NoteGrU (note group stem up) if there is a **stem**, and close to the left of the down tip of the **stem**, there is a NoteHead."

7.7 A grammar for primitive assembly

In the three existing grammar-based OMR systems, the grammar is used to perform more than one task. Consequently the grammar methodologies used are sophisticated and complex to implement. For this work we only require the grammar to assemble the primitive shapes. Can a simpler grammar-based solution be found for this task?

The fundamental form of a grammar—the string grammar—is simple to implement, but this type of grammar is insufficient to parse systems where tokens are two-dimensional in layout. This is why Coüasnon *et al.* augmented the basic DCG notation with a positional operator and factorisation. The extensions are attractive, although it is the exten-

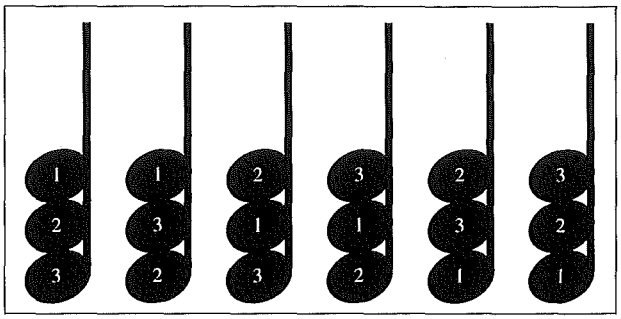


Figure 7.10: There is more than one way to assemble the note heads in a chord.

sions themselves that motivated the move to λProlog to implement the system, and even then the authors reported that developing the parser was a difficult task.

An alternative approach is to replace the one-dimensional list of tokens with a two-dimensional data-structure, such as a tree or a graph. This is the route taken by Fahmy and Blostein, and for many other structured document image analysis problems. Unfortunately, here too are reports of the difficulty in developing parsers [San92].

We therefore return to the basic form of grammar and adopt a different route, which is to change the *list* of tokens to a *bag* of tokens. Because a bag is also a one-dimensional data-structure, the development of a parser is kept simple.

The bag data-structure is a cross between a list and a set. For a bag, the order of objects does not matter, and the same element can appear more than once. This is straightforward to implement in Prolog as a predicate that extracts elements from a list, with unrestricted backtracking. A modified DCG notation is therefore selected as the implementation medium for our grammar-based knowledge representation of primitive assembly.

In terms of run-time complexity, the introduction of a bag data-structure is potentially dangerous since a grammar can now specify combinatorially large search spaces. For example, if a grammar based on a bag of primitives does not specify the order in which note heads next to a stem are assembled, then for every chord in a piece of music with n note heads, there are $n!$ combinations of note heads that satisfy the grammar. This point is illustrated in Figure 7.10, where six different orderings of three note heads assemble to form the same crotchet chord. In this situation the control of access to the knowledge-base is *under-specified.*

Fortunately this danger can be contained because, in this application, we are both

Translator	Number of lines	Number of characters
Definite Clause Grammar	120	3719
Bagged Clause Grammar	216	10402

Table 7.2: Comparison of the original DCG implementation with the modified BCG implementation.

the specialist in music notation and computer knowledge representation design. Being knowledgeable about music notation we understand the geometric requirements on note heads in a chord, and being knowledgeable about computer algorithms we can guard against under-specification by embedding constraints and controlling backtracking in the grammar.

7.7.1 Implementation

An existing implementation of a DCG translator (in Prolog) was adapted to use bags rather than lists.[2] Statistics on the size of the original DCG translator, and the modified translator (dubbed a Bagged Clause Grammar, or BCG) are shown in Table 7.2.

The extension approximately doubles the length of the implementation. Most predicates carry extra arguments, and of the 96 extra lines added, 31 lines (2087 characters) support predicates specific to our primitive assembly task.

Supporting predicates are:

`page_info(LinkName)`

This predicate forms a link between the grammar and pre-processed information about the scanned page. Using this link a production rule can access, for example, scan resolution, staff system information (local or global), staff information (local or global), staff line thickness (local or global), and so on.

`prim_present(Prim,Xl,Yt,Xr,Yb,CheckType)`

This predicate checks for the existence of the named primitive in the specified rectangle. There are three types of check: at least one `corner` of a primitive of the named type falls inside the specified rectangle; a primitive of the named type fits completely `inside` the specified rectangle; and a primitive of the named type and

[2]The DCG translator modified was from the source distribution of NU-Prolog written by the Department of Computer Science, University of Melbourne, Parkville, Victoria, Australia.

the specified rectangle `intersect`.

`prim_not_present(Prim,Xl,Yt,Xr,Yb,CheckType)`

This predicate computes the negation of `prim_present`.

`prim_present_static(Prim,Xl,Yt,Xr,Yb,CheckType)`

As the assembly process proceeds, the primitives used to assemble a particular instance of a musical feature are removed. Thus the bag of primitives used by the predicate `prim_present` changes dynamically. When checking for primitives, this is not always desirable. An alternative is to use the predicate `prim_present_static` which performs the same function, except a static snapshot of the bag at the very start of the assembly process is used.

`prim_not_present_static(Prim,Xl,Yt,Xr,Yb,CheckType)`

This predicate computes the negation of `prim_present_static`.

`store_attribute(AttName,AttValue)`

It is possible to annotate the derivation tree of an assembled musical feature. Using this predicate in a production rule stores the named field and value at the corresponding node in the derivation tree. Permissible values are integers, floating-point numbers, and strings.

7.7.2 Examples

To illustrate the techniques used in primitive assembly using a BCG, let us now look at some examples taken from the grammar for CMN assembly.

Figure 7.11 shows example notes from a collection of musical features we will call simple up notes. In Figure 7.12, BCG rules are given that represent such notes. The example has been simplified to focus attention on the structure of the rules. The first production rule (starting at line 1) picks a vertical line from the bag, and looks for suitable tails and note heads. The second production rule (starting at line 6) allows a vertical stack of note heads to be associated with the stem. The `note_head_within_up` production (starting at line 12) checks to see if a selected note head is close enough to the stem. The note head to check is provided by the `note_head` production (starting at line 16) which also discovers any durational dots to the right of the note head using `opt_dur_dots` (starting at line 22),

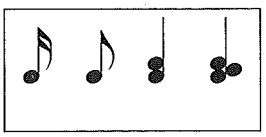


Figure 7.11: Examples of simple up notes.

```
1  simple_note_up
2           ==> [(vertical_line,_,Sxl,Syt,Sxr,Syb,_)],
3               opt_tails_up(Sxl,Syt,Syb,NoTails),
4               note_heads_up(Sxl,Syt,Syb,NoTails).
5
6  note_heads_up(Sxl,Syt,Syb,NoHalves)
7           ==> note_head_within_up(Sxl,Syt,Syb,NoHalves),
8               note_heads_up(Sxl,Syt,Syb,NoHalves).
9  note_heads_up(Sxl,Syt,Syb,NoHalves)
10          ==> note_head_within_up(Sxl,Syt,Syb,NoHalves).
11
12 note_head_within_up(Sxl,Syt,Syb,NoHalves)
13          ==> note_head(Syt,Syb,NoHalves,Nxl,Nyt,Nxr,Nyb),
14              { % note head must be close to stem (to the left) }.
15
16 note_head(Syt,Syb,NoHalves)
17          ==> [(full_note_head,_,Nxl,Nyt,Nxr,Nyb,_)],
18              { % retrieve appropriate StaffGap for note head },
19                opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots),
20              { % store note head duration etc }.
21
22 opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots)
23          ==> opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,0,NoDots).
24 opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDotsIn,NoDotsOut)
25          ==> [(dot,_,Dxl,Dyt,Dxr,Dyb,_)],
26              { % check dot close enough to note head },
27              { IncNoDots is NoDotsIn + 1 },
28                opt_dur_dots(Dxr,StaffGap,Nyt,Nyb,IncNoDots,NoDotsOut).
29 opt_dur_dots(_,_,_,_,NoDots,NoDots)
30          ==> []. % epsilon
31
32 % opt_tails_up is similar to opt_dur_dots
```

Figure 7.12: Edited BCG production rules that described simple up notes consisting of note heads, a stem, dots and tails.

and stores the duration of the note in the derivation tree. The production rule for tails (`opt_tails_up`) is identical in structure to `opt_dur_dots`, except a (possibly non-existent) vertical stack of tails is built, rather than a (possibly non-existent) horizontal line of dots.

In Figure 7.13 an excerpt of the simplified BCG of Figure 7.12 (lines 12–30) is shown in full detail.

Constraints that were explained in English in the simplified figure, are defined by boolean expressions. For example, on line 3 the centre of the note head picked from the bag is constrained to fall between the top and the bottom of the stem. If this condition

```
note_head_within_up(Sxl,Syt,Syb,NoHalves)
         ==> note_head(Syt,Syb,NoHalves,"up",Nxl,Nyt,Nxr,Nyb),
             { mid(Nyt,Nyb,Nym), (Nym>=Syt) and  (Nym=<Syb), % within
               dim(Nxl,Nxr,Nxd), mid(Nxl,Nxr,Nxm),
               Wxr is Sxl, Wxl is Sxl - (Nxd//4),
               % note head must be close to stem (to the left)
               (Nxr>=Wxl) and (Nxm=<Wxr) },
             !. % committed choice


note_head(Syt,Syb,NoHalves,NoteDir)
         ==> [(full_note_head,_,Nxl,Nyt,Nxr,Nyb,_)], page_info(PageInfo),

             % retrieve appropriate StaffGap for note head
             { mid(Nyt,Nyb,Nym), mid(Nxl,Nxr,Nxm),
               max_gap_height(PageInfo,Nym,StaffGap) },

             opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots),
             { cmn_time_unit_duration(8,4,NoHalves,NoDots,NoteHeadDur) },
             { pi_staff_systems(PageInfo,StaffSystems),
               closest_staff(StaffSystems,Nxm,Nym,StaffSystemNo,StaffNo) },

             % store note head duration etc
             store_attribute("time_atomic","true"),
             store_attribute("time_event_dur",NoteHeadDur),
             store_attribute("note_dir",NoteDir),
             store_attribute("staff_system_no",StaffSystemNo),
             store_attribute("staff_no",StaffNo),
             store_attribute("x_hotspot",Nxm),
             store_attribute("y_hotspot",Nym).


opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots)
         ==> opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,0,NoDots).

opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDotsIn,NoDotsOut)
         ==> [(dot,_,Dxl,Dyt,Dxr,Dyb,_)],

             % check dot close enough to note head
             { (Dxl>Nxr) and (Dxl<Nxr+StaffGap), % within
               mid(Dyt,Dyb,Dym), HalfStaffGap is StaffGap // 2,
               (Dym>Nyt-HalfStaffGap) and (Dym<Nyb+HalfStaffGap) },

             { IncNoDots is NoDotsIn + 1 },
             opt_dur_dots(Dxr,StaffGap,Nyt,Nyb,IncNoDots,NoDotsOut).

opt_dur_dots(_,_,_,_,NoDots,NoDots)
         ==> []. % epsilon
```

Figure 7.13: Unedited BCG production rules for simple up notes.

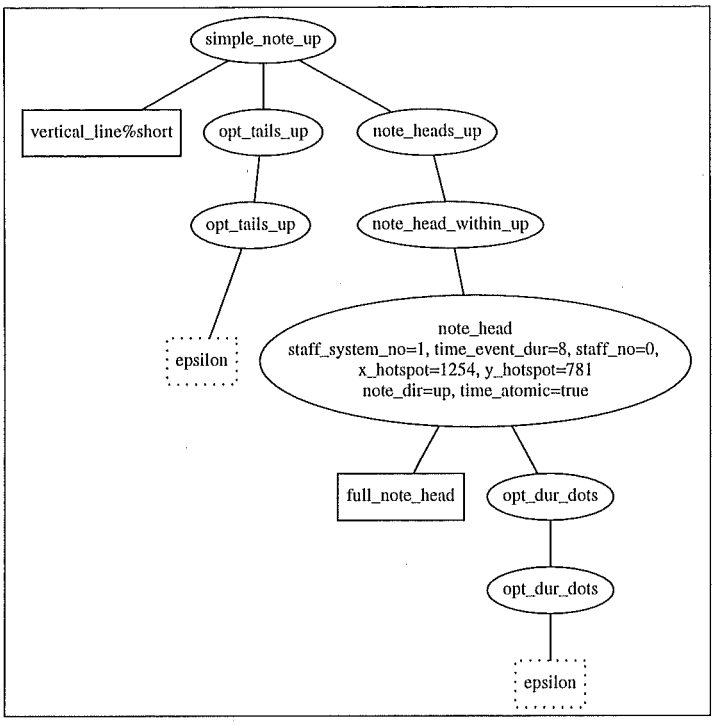


Figure 7.14: The annotated derivation tree for a crotchet note.

is not met, then the note head is discarded, and another is chosen. Note also the use of the cut operator ('!') on line 8, which controls back-tracking. Once a note head has been found that binds to a stem, there is no need to go back to try and start a vertical stack of note heads with a different note head. Thus the undesired combinatorial situation illustrated in Figure 7.10 is prevented.

Figure 7.13 also shows the use of supporting predicates specific to OMR. The `mid` and `dim` predicates respectively calculate the mid-point and the distance between two numbers. On line 12, the `page_info` predicate is used to form a link to the page specific data. This link is then used on line 16 to obtain the value of the staff gap resulting from the largest staff that can place musical features on the pixel line in the image specified by `Nym`. Lines 24–30 store information in the derivation tree for future use by the musical semantics stage of the OMR system—how these values are used is explained in the next chapter. The derivation tree from a crotchet note is shown in Figure 7.14.

To see an example of the `prim_present` predicate, let us study the BCG production rule for a sharp, which is shown in Figure 7.15. Since a sharp can form part of a key

```
 1  sharp
 2     ==> [(sharp,_,Sxl,Syt,Sxr,Syb,_)],  page_info(PageInfo),
 3         { pi_staff_systems(PageInfo,StaffSystems),
 4           intersects_staff(StaffSystems, (Sxl,Syt,Sxr,Syb), StaffSystemNo,
 5                            StaffNo, StaffRect,StaffInfo),
 6           staff_info_staff_height(StaffInfo,StaffHeight),
 7           SharpGap is StaffHeight // 2,
 8           Cxl is Sxr, Cxr is Sxr + SharpGap, Cyt is Syt, Cyb is Syb },
 9         note_head_present(Cxl,Cyt,Cxr,Cyb),
10         { mid(Sxl,Sxr,XHotSpot), mid(Syt,Syb,YHotSpot) },
11         store_attribute("secondary_atomic","true"),
12         store_attribute("acc_type","sharp"),
13         store_attribute("staff_system_no",StaffSystemNo),
14         store_attribute("staff_no",StaffNo),
15         store_attribute("x_hotspot",XHotSpot),
16         store_attribute("y_hotspot",YHotSpot).
17
18  note_head_present(Cxl,Cyt,Cxr,Cyb)
19     ==> prim_present_static(full_note_head,Cxl,Cyt,Cxr,Cyb,intersect).
20  note_head_present(Cxl,Cyt,Cxr,Cyb)
21         prim_present_static(hollow_note_head,Cxl,Cyt,Cxr,Cyb,intersect).
22  note_head_present(Cxl,Cyt,Cxr,Cyb)
23         prim_present_static(semibreve,Cxl,Cyt,Cxr,Cyb,intersect).
```

Figure 7.15: The BCG production rule for a sharp.

signature or appear on its own, it is necessary to distinguish between the two situations. This is accomplished by lines 18–23, which confirm that the sharp under consideration appears in isolation by checking to see if there is a note head immediately to the right of the sharp. The static version of the predicate is used because the assembly of notes occurs before accidentals, and therefore note heads are no longer in the dynamically changing bag.

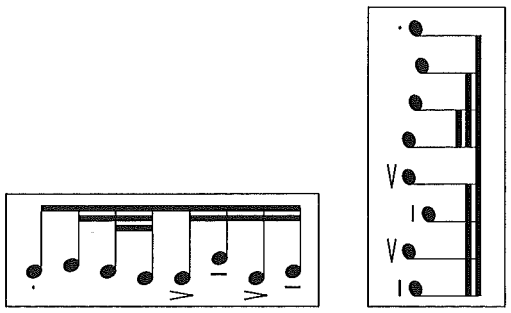
A similar use of `prim_not_present_static` is made in the production rule for a single bar line, shown in Figure 7.16. For a vertical line to be "assembled" into a bar line, its length must be at least 80% of the minimum staff height operating in the area of the vertical line (line 6), it must intersect a staff (line 9), and the top of the vertical line must be located close to the top of the staff it intersects with (lines 12–17). Alone, however, this is an insufficient specification for a single bar line, since it is conceivable that a note might be drawn in such a way that its stem is long enough to be a bar line, and the top of the note stem stops near the top of the staff. The discrepancy is resolved by using `prim_not_present_static` to rule out potential bar lines that have note heads too close to the vertical line (line 20–24).

As a final example, an informal explanation of the production rules for beamed notes is given since its coding is prohibitively long for presentation in this chapter.

A beamed note can be a complex, nested structure, such as the example shown in

```
1  single_bar_line
2     ==> [(vertical_line,_,Bxl,Byt,Bxr,Byb,_)], page_info(PageInfo),
3         { dim(Byt,Byb,Byd), mid(Byt,Byb,Bym),
4           min_staff_height(PageInfo,Bym,StaffHeight),
5           % bar line 80% of staff height
6           Byd > (80 * StaffHeight // 100)},
7         { pi_staff_systems(PageInfo,StaffSystems),
8           % bar line intersects staff
9           intersects_staff(StaffSystems,(Bxl,Byt,Bxr,Byb),StaffSystemNo,StaffNo, _,_),
10
11          % top of vertical line start at same height of staff
12          retrieve_staff_from_staff_systems(StaffSystems,StaffSystemNo,StaffNo,Staff),
13          staff_ytop(Staff,Syt),
14          staff_staff_info(Staff,StaffInfo),
15          staff_info_gap_height(StaffInfo,StaffGap),
16          BarGapTol is StaffGap // 2,
17          (Byt >= (Syt - BarGapTol)) and (Byt =< (Syt + BarGapTol)) }, % within
18
19         % no note heads close by
20         { BarNotPresTol is StaffGap,
21           NPxl is Bxl - BarNotPresTol, NPxr is Bxr + BarNotPresTol,
22           NPyt is Byt - BarNotPresTol, NPyb is Byb + BarNotPresTol },
23         prim_not_present_static(full_note_head,NPxl,NPyt,NPxr,NPyb,intersect),
24         prim_not_present_static(hollow_note_head,NPxl,NPyt,NPxr,NPyb,intersect),
25
26         store_attribute("measure_atomic","true"),
27         store_attribute("staff_system_no",StaffSystemNo).
```

Figure 7.16: The BCG production rule for a single bar line.



(a) (b)

Figure 7.17: A different way of looking at beamed notes (a) normally (b) side-on.

Figure 7.17a. Writing production rules to represent this assembly is, perhaps, a daunting prospect. However, if tipped on its side, a beamed note—such as the example shown in Figure 7.17b—can been seen as an abstract version of a block-nested programming language: beams correspond to nested blocks, and stems correspond to statements. Developing grammar rules for a block-nested programming language is a familiar task—often included in a compiler assignment for an undergraduate course. The task of assembling beamed notes is no different, with the same pattern of production rules forming the main structure to this part of the grammar.

Of course the comparison is not exact, the main difference being the structure of a beamed note permits stems to be placed on both sides of the primary beam. Fortunately, this is a superficial difference that can be encoded by developing two sets of rules in parallel that mirror one another in the structures they describe.

7.8 Evaluation

In evaluating the success of this stage of the OMR system, two important considerations are *accuracy* and *speed.* First we must quantify how accurately the system assembles primitives into musical features, and second, we are concerned with the speed of this operation, since a 100% accurate system is of little use if it takes an impractical amount of time.

Within the basic strategy of a BCG, the search space for assembly can be reduced by filtering the bag of primitives at various points. For example, once a vertical line has been picked from the bag of primitives to form a note stem, the remaining primitives can be filtered so only those close to the selected vertical line are left, thus reducing the number of checks necessary to assemble a note. This leads to faster assembly times, although an over-pruned search space reduces accuracy. Four strategies within a BCG framework are to work with a bag of primitives that is:

- unfiltered,
- filtered with respect to the staff systems,
- filtered by embedded production rules, or
- filtered with respect to the staff systems, and then by embedded production rules.

When filtering primitives with respect to a staff system, the region to process is formed by extending the top of the current staff system to the bottom of the staff system above, and the bottom of the current staff system to the top of the staff system below. The primitives are then filtered, based on this rectangular region, and those that fall inside the region are processed by the grammar. The filtered primitives that remain after this application of the grammar are then merged with the primitives that fell outside the rectangular region, and the whole process is repeated again for the next staff system. If no staff system exists above or below the current staff system, then the appropriate edge of the page is used as the limit.

Below, we compare the speed and accuracy of these four strategies applied to examples of CMN taken from the corpus. The claim that basing this stage of an OMR system on a formal knowledge representation scheme leads to a flexible, extensible system, is supported by statistics from the development of a grammar expressing the assembly requirements for plainsong notation.

7.8.1 Common music notation

Table 7.3 lists the musical features currently supported by the grammar for CMN. The set of assembled musical features covers the core of this music notation. Extending the grammar to include additional musical features is straightforward. In most cases repeating an already existing pattern of non-terminals and constraints is all that is required.

Table 7.4 presents statistics on the length of the implemented grammar. Of the 751 lines written, over half calculate constraints and store attributes in the derivation tree which, although tedious to write, is a simple task. The remaining 281 lines encode the structure of the assembly process.

Accuracy

In this section the assembly process is evaluated using "Promenade," "Bonita," and "Big My Secret." A more comprehensive account is presented in Chapter 9, when a complete OMR system is processed.

The notational layout in "Promenade" (Figure 7.18) is complex. Not only does it include a varied assortment of clefs, time signatures, accidentals, rests, and notes, it also

Musical feature group	Additional comments
Clefs	Treble clef, bass clef, and alto clef.
Time signatures	Any fraction using the digits [1..16].
Key signatures	7 with sharps, and 7 with flats.
Simple notes	Filled-in or hollow note heads, tails, and durational dots, with optional staccato, stress, and accent marks.
Beamed notes	Includes beamed notes with stems drawn on both sides (normally done when a beamed note crosses between two staves).
Stem sharing notes	Simple up note and down note combined.
Rests	Semiquaver, quaver, crotchet, minim, semibreve (including duration dots).
Accidentals	Sharp, flat, natural, double sharp, and double flat.
Slurs/ties	No distinction is made between the two musical features at this point in the OMR system.
Bar lines	Single line and double line.
Text	Text characters are assembled into words.

Table 7.3: The musical features supported by the current grammar for CMN.

Total Number of lines of code	751
Number of lines of code involved with constraints	383
Number of lines of code involved with storing attributes	87

Table 7.4: Statistical information about the implemented grammar for CMN assembly.

switches to split voice layout at numerous points throughout the work. The piece, therefore, makes a good test case for the accuracy of primitive assembly.

In addition to this notational complexity, the quality of the engraving and printing is high, so assembly statistics are not affected by defects in earlier OMR stages. Adequate space has been given to individual shapes so, for example, no accidentals or slurs touch note heads, and each shape is boldly rendered in the original, making fragmentation in the scanned image unlikely. This leads to an extremely high recognition rate in the primitive detection stage of the OMR system. After the removal of text by the OCR stage, *all* primitive shapes are correctly identified. For two flats and one crotchet rest in the image, the match is uncertain, but no competing primitives are classified for these shapes.

All four strategies produce identical accuracy results, indicating that the filtering

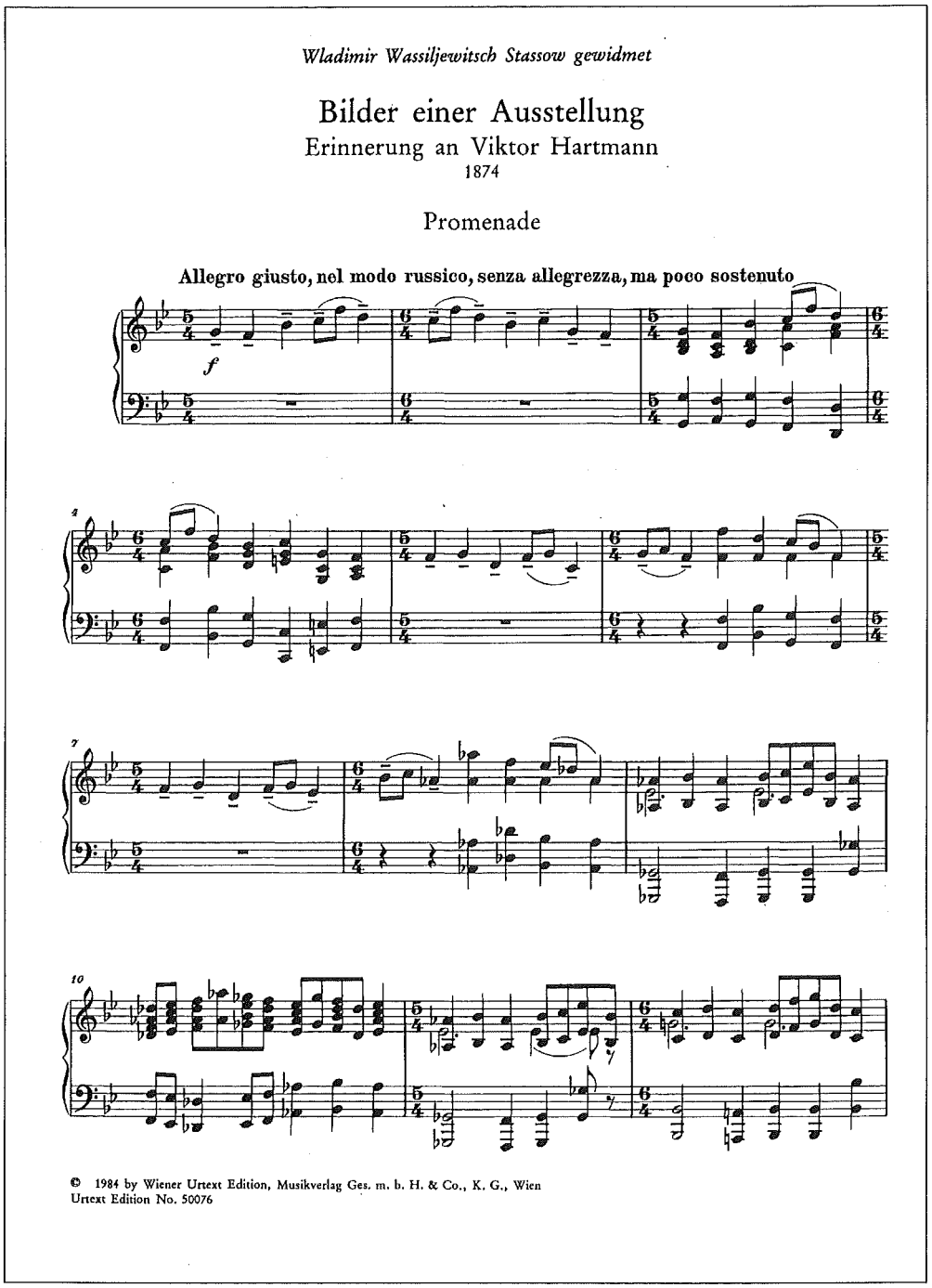


Figure 7.18: "Promenade" is an example of CMN that includes complex layout.

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note head	5	5	0
Note head with stem	255	255	0
Tail with stem	2	2	0
Note with beam	44	44	0
Nested beams	0	0	0
Bass clef curl with two dots	4	4	0
Accidentals in key signature	16	16	0
Total	326	326	0

Table 7.5: Statistics for the assembly of true primitives in "Promenade."

rules are sufficiently generous in the regions used to retain primitives. Table 7.5 gives a breakdown of the accuracy rate for the example image based on the type of assembly operations. For this image, perfect assembly results are achieved. Indeed, accuracy rates close to 100% are typical for this stage of the system because the structural information in musical feature taxonomy is high. This makes it easy to devise robust rules for assembly.

Most errors in the assembly stage can be traced back to mistakes during primitive detection. Tables 7.6 and 7.7 give a similar break down of assembly operations for the first two pages of two additional works ("Big My Secret" and "Bonita"). The quality of layout in "Bonita" is substantially lower than the other two works, and the primitive detection stage fails to classify some of the primitive shapes—most notably 35% of flats and 63% of dots are left unclassified. Tables 7.8 and 7.9 show the same break down of assembly operations for the same two pieces of music, only this time the expected number of assembly operations is based on the primitives the computer detected, rather than the true primitives that exist in the work.

Assembly for "Bonita" is now perfect, and "Big My Secret" fails only once in the assembly of a note stem with a beam. Figure 7.19 shows the excerpt where the mistake occurs. The OCR stage failed to classify the fingering information digit 5, and the robust rule for detecting vertical lines—that allows for small breakages to occur—included the digit 5 as part of the stem below it. Consequently, the top of the detected vertical line is too high to be considered part of the beamed note.

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note head	6	6	0
Note head with stem	478	478	0
Note with beam	88	88	0
Nested beams	4	4	0
Bass clef curl with two dots	2	2	0
Accidentals in key signature	24	18	0
Total	602	596	0

Table 7.6: Statistics for the assembly of *true* primitives in "Bonita."

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note head	22	22	0
Note head with stem	483	483	0
Note with beam	411	410	0
Nested beams	99	99	0
Bass clef curl with two dots	8	8	0
Accidentals in key signature	32	31	0
Total	1055	1053	0

Table 7.7: Statistics for the assembly of *true* primitives in "Big My Secret."

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note head	6	6	0
Note head with stem	478	478	0
Note with beam	88	88	0
Nested beams	4	4	0
Bass clef curl with two dots	2	2	0
Accidentals in key signature	18	18	0
Total	596	596	0

Table 7.8: Statistics for the assembly of *detected* primitives in "Bonita."

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note head	22	22	0
Note head with stem	483	483	0
Note with beam	411	410	0
Nested beams	99	99	0
Bass clef curl with two dots	8	8	0
Accidentals in key signature	31	31	0
Total	1054	1053	0

Table 7.9: Statistics for the assembly of *detected* primitives in "Big My Secret."

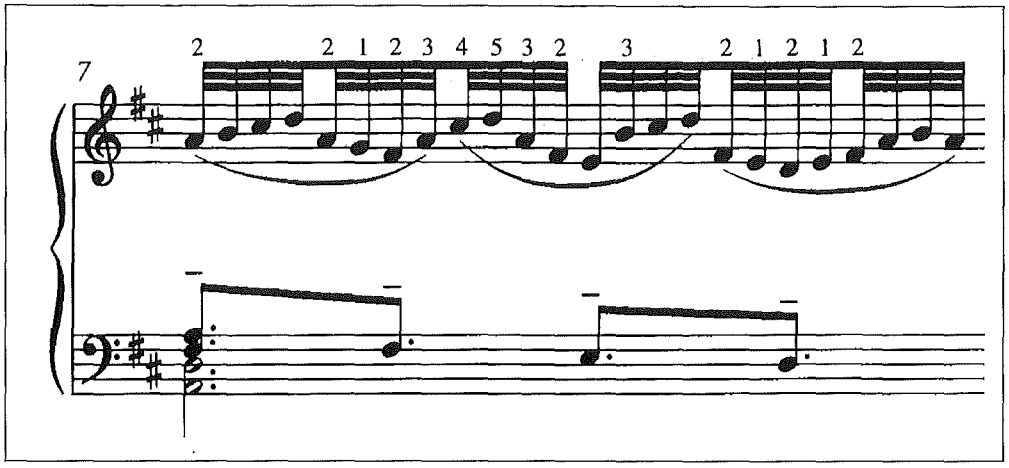


Figure 7.19: The excerpt of "Big My Secret" where a failure in the OCR stage to detect the digit 5 leads to a vertical line being detected that extends too far above the beam.

Unfiltered	1830 secs	
Staff system filtered	454 secs	
Production filtered	54 secs	
Staff system and production filtered	31 secs	

Figure 7.20: Comparison of timing information for primitives assembly using various filtering strategies.

Processor time

Figure 7.20 shows the average computational cost of the four different strategies when applied to the selected works used above: "Promenade," "Bonita," and "Big My Secret."

Successive filtering strategies drastically improve performance. The unfiltered strategy takes over 30 minutes per page due to the high number of comparisons made during assembly. To assemble a "basic note" the grammar picks a vertical line from the bag, and then proceeds to compare every note head on the page to this vertical line to see if it is close enough to count as part of the basic note. The number of comparisons is far worse for beamed notes. For some forms of notation it might be necessary to check primitives that are physically far apart, thus necessitating this computationally expensive strategy, but this is not so for CMN. Filtering primitives to those located on or near a staff system results in a speed up factor of four.

Even faster times result from embedding filtering operations in the grammar. Em-

Total Number of lines of code	197
Number of lines of code involved with constraints	91
Number of lines of code involved with storing attributes	24

Table 7.10: Statistical information about the implemented grammar for plainsong notation assembly.

bedded rules that filter a page of primitives result in a speed up factor of 34, and embedded rules that filter primitives already filtered by staff system result in a speed up factor of 59. Only two filtering operations were embedded to achieve this improvement. The first filtering rule is located after a stem is picked from the bag, and restricts primitives to those close to the chosen stem. The second filtering rule is located after the main beam to a beamed note is picked from the bag, and restricts primitives to those close to the chosen beam.

7.8.2 Plainsong notation

To demonstrate the flexibility of the BCG approach to musical feature taxonomy, a second grammar was written to assemble the primitive shapes of plainsong notation. Table 7.10 presents statistics on the length of the implemented grammar. The grammar for plainsong notation is smaller than that for CMN because the assembly requirements are less intricate for this notation. The proportion of code for constraints and storing attributes, however, is similar to CMN at slightly over half the total number of lines.

Figure 7.21 shows the example image used in the previous chapter for plainsong notation primitive detection. After processing by the assembly stage, a graphically "cleaned-up" version of the piece can be automatically constructed in a drawing package. This has been done in Figure 7.22. Assembled primitives that form one musical feature are grouped together to form a single object which can then be modified in the drawing package if desired. The automatically constructed picture shown is a crude implementation of editorial enhancement which was presented in the introduction of this thesis as one of the applications for OMR. It is possible to refine the procedure that constructs the picture to automatically produce a more aesthetically pleasing version.

Like assembly for CMN, assembly for plainsong notation is characterised by extremely high accuracy rates. In Figure 7.22 only two mistakes have been made, and both of these are due to errors in the primitive detection stage. Because no PRIMELA descrip-

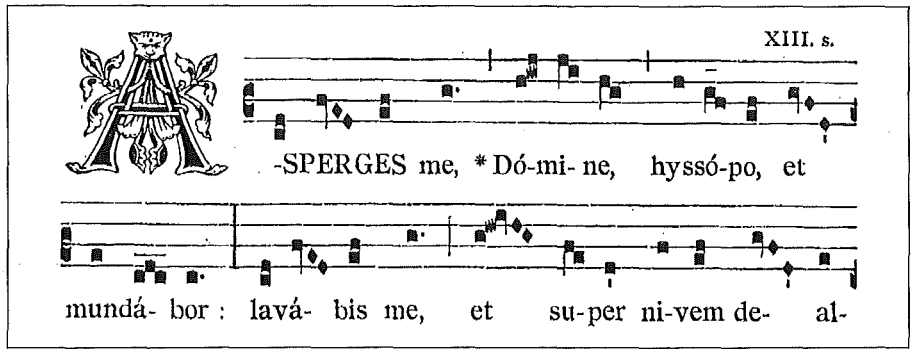


Figure 7.21: The example excerpt of plainsong notation used to demonstrate primitive detection.

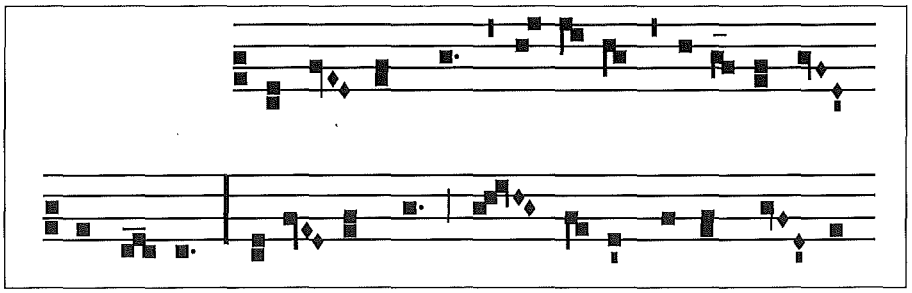


Figure 7.22: A graphically cleaned-up version of the example excerpt automatically constructed in a drawing package.

tion was written for the saw-toothed shape, the first occurrence of this primitive is left unclassified, and consequently not assembled. The second occurrence is mistaken for a rectangle, and is therefore incorrectly assembled.

Chapter 8

Musical semantics

The final stage in the OMR process is to derive the musical semantics of the piece. Much processing has been undertaken to reach this point: starting with a scanned image, the OMR system has located the staff lines, isolated the objects from the staff lines, optionally enhanced the image, detected the primitive shapes contained in the isolated objects, and assembled these shapes into musical features. Finally the system is in a position to interpret the musical content.

Since there is no standard file format for music representation, we regard the data structure generated by the musical semantics stage as the end point of the system. From this data structure, files can be produced conforming to any desired musical file format.

Different formats place different requirements on the data structure. For instance, to generate a MIDI[1] [Rum90] file for audio playback, the ultimate pitches and durations of notes must be specified. Calculating this requires the combination of clef, key signature, time signature, accidentals, and ties. Other musical features such as bar lines and lyrics are ignored. Conversely, the Tilia file format used by the LIME music notation editor [HB93] retains much of the structure in music notation, requiring the explicit specification of clefs, key signatures, time signatures, accidentals, ties, bar lines, and lyrics. The data structure used for this work must be capable of supporting this diverse range of musical file formats.

Explanations of the musical semantics stage of OMR systems are rare in the literature. The reasons are two-fold: first, many systems do not take the OMR process to its ultimate conclusion, ending instead with the graphical classification of musical features;

[1]MIDI is an abbreviation for Musical Instrument Digital Interface.

and second, even when a project does implement this stage, the operations are too specific to the particular implementation—due to the choice of algorithms for previous stages—to make publication worthwhile.

In this chapter, therefore, we start by describing the implementation of the musical semantics stage of an OMR system in broad terms, showing how existing explanations fit this general description. Next, we explain the design of the musical semantics component for this work, where once again extensibility is the key issue. The chapter ends with an evaluation of the work.

The design is centred around two constructs, which we have called *time-threads* and *structured lists*. Time-threads form a two-dimensional lattice-like structure, linking all musical events together, where a musical event is an indivisible entity within the context of musical semantics, such as a note head, rest, or key signature. Threads run horizontally and vertically. For CMN, this equates to linking musical events sequentially and concurrently. A structured list is built on top of each horizontal thread, and provides multiple *levels* for traversal. As a brief explanation, when traversed at one level, all the musical events for a staff are grouped together, and can therefore be covered in one pass. This is a convenient level to apply the effects of musical events such as clefs and key signatures to notes. Alternatively, when traversed at a different level, the same events are grouped together as individual bars, which is a convenient level to apply the effects of, for example, accidentals on notes. A more detailed explanation of structured lists is given in Section 8.2.3.

In previous chapters the work performed is often applicable to other document image analysis problems—for instance, the combination of PRIMELA and a bagged clause grammar (BCG) could also be used to recognise circuit diagrams. The musical semantics stage is considerably less general, as the dependency on time-threads restricts the class of problems applicable to those where time is conveyed using displacement in the x and y axes.

8.1 Interpreting spatial relationships

Before we explain the musical semantics component designed for this work, we will first describe the required operations in general terms.

The main requirement of this stage is to combine the graphically recognised musical features with the staff systems to produce a musical data structure representing the

meaning of the scanned image. This is accomplished by interpreting the spatial relationships found in the image. In applications like OCR this is a simple (almost trivial) step since the layout is predominately one-dimensional. For music however, the layout is much more complex. Positional information is extremely important. The same graphical shape can mean different things in different situations. For instance, to determine if an object between two notes is a slur or a tie, the pitch of the two notes must be considered. Also, establishing the association of "free-floating" objects, such as a hairpin crescendo, with the appropriate notes is an important task in understanding music.

Broadly speaking, in an OMR system where the musical semantics stage *follows* primitive assembly, this phase of an OMR system consists of (possibly) multiple passes over a graph-like data structure, creating links, deleting links, and modifying the attributes stored at nodes due to the effect of one musical feature, such as a key signature, on other musical features, such as notes. In an OMR system where the extraction of musical semantics is implemented *simultaneously* with other stages, these operations are dispersed throughout the system, and the graph is built incrementally.

In a prototype system by the author [Bai91], the list of graphically classified musical features is translated into a graph-like musical data-structure using multiple passes. First, musical features are grouped according to staff systems, staves, and then bars. Next basic pitches,[2] and basic durations[3] are calculated for all notes and rests, which are then modified by the effects of time signatures, key signatures, accidentals, and clefs—in that order.

The construction of the graph is explicit in Fahmy and Blostein's system [FB91, FB92] with their "Build-Constrain-Incorporate" strategy forming links, deleting links, and modifying node attributes. The overlap between primitive assembly and musical semantics means the graph structure is built incrementally, and also includes the construction of musical features. Reed's system [Ree95] is similar, with the musical semantics stage loosely based on the graph grammar approach cultivated by Fahmy and Blostein.

In Andronico and Ciampa's description of their system [AC82] it is explained how to traverse their graph-like data structure of recognised musical features to produce the desired musical semantics. Predefined variables act as carriers for specific musical fea-

[2]Pitch based solely on the position of the event with respect to the staff. For instance, '1' represents the bottom line of a staff, '2' represents the first staff gap above this, and so on.

[3]Duration based solely on the graphical primitives that constitute the musical feature. For instance, '4' represents a crotchet, '8' represents a quaver, and so on.

tures, maintaining the relevant value during the scope of the musical feature's influence. For instance, an array is kept for accidentals, where each entry corresponds to one pitch. The pitch of subsequent notes are altered by the corresponding entries in this array; when a bar line is encountered, the array is reset to zero. Due to the simple set of CMN recognised, the traversal could be implemented as a single pass. In fact, after implementation considerations, the authors chose to merge this step with the recognition procedure, invoking the musical semantics routine after each classification of an unknown object.

8.2 The design of the musical semantics stage

The design of this stage of the OMR system is based on a pre-constructed graph that the system automatically builds by combining the assembled musical features with the detected staff systems. To meet the requirement of extensibility, the traversal and modification of the pre-constructed graph are programmable components of the system, controlled by the configurer. Thus in CMN, for example, the effect of a key signature can be applied by supplying a routine that traverses the part of the graph representing the staff that the key signature is from, modifying the pitch of any note head encountered whose pitch matches that of an accidental in the key signature. A subsequent routine applying the effect of accidentals on note heads within a bar ensures the correct scoping of key signatures and accidentals.

Since there is no recognised standard file format for music representation, the graph resulting from the application of the programmable component is viewed as the musical interpretation of the scanned image for this project. Additional routines can be provided by the configurer to traverse this structure generating files conforming to particular file formats, such as MIDI, CSound, Tilia, Finale, and DARMS.[4]

8.2.1 The pre-constructed graph

Based on the detected staff systems and assembled musical features, a skeleton graph is constructed at the beginning of the programmable musical semantics stage. A node in the graph represents a *musical event*—an entity that is indivisible with respect to its musical semantics. A musical event can be a primitive shape, such as a note head, or a musical

[4]DARMS is an abbreviation for Digital Alternate Representation of Musical Score.

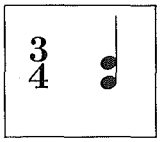


Figure 8.1: An example musical feature from each atomic category.

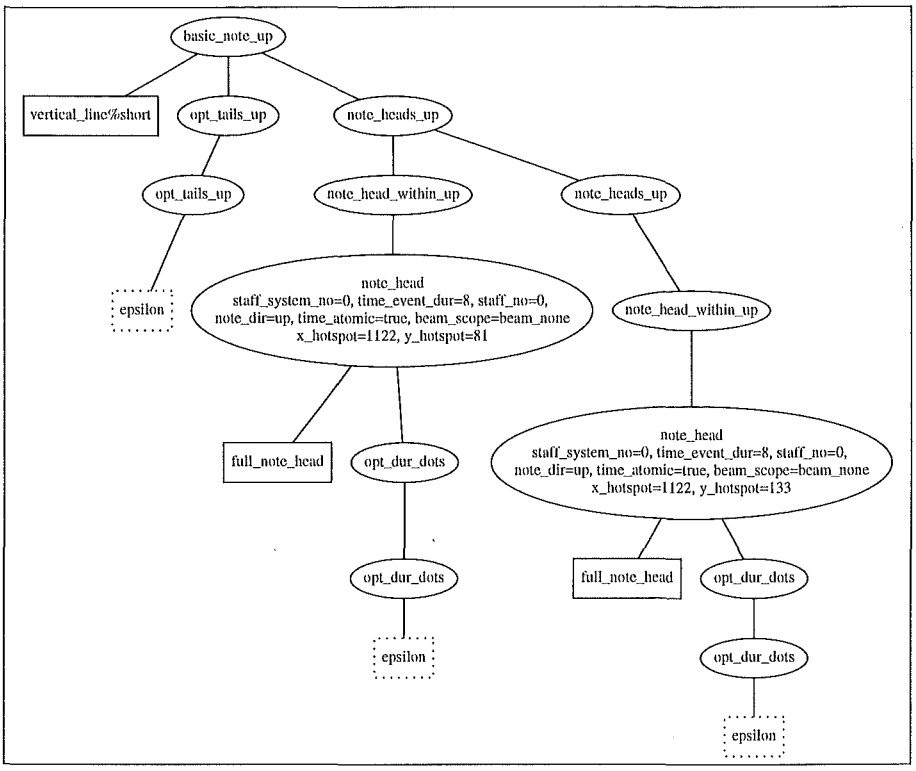
feature, such as a bass clef. In fact, a musical event can be a partially assembled musical feature, although it is rare to find an example between the two extremes.

Assembled constructs that are musical events are specified during the primitive assembly phase. The annotated attribute field `time_atomic` or `secondary_atomic` stored at a node of the derivation tree defines the root of a sub-tree corresponding to the structure that is a musical event. The two categories exist to distinguish between musical events that directly involve duration (notes and rests) and those that do not—effectively all other musical events. Although musical events such as *ritardando* and *presto* affect duration, they do so indirectly, and are therefore classified as `secondary_atomic`.

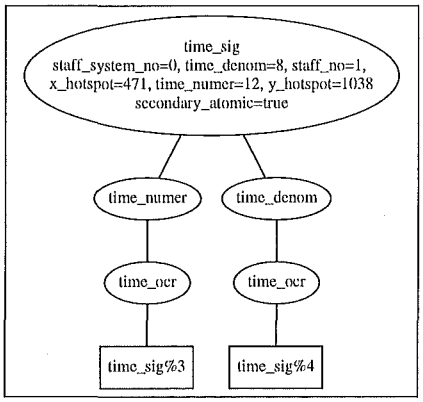
In Figure 8.1 an example of each atomic category is shown. The derivation tree for the crotchet chord (Figure 8.2a) is annotated with `time_atomic` twice—once for each note head—seen in the two large elliptical nodes of the tree, whereas the derivation tree for the time signature (Figure 8.2b) is annotated with a single `secondary_atomic` attribute at the top node, since this musical feature has no duration of its own.

Strictly speaking, the annotation for `time_atomic` and `secondary_atomic` in the primitive assembly phase is a breach of our proposed framework, where all the stages to OMR are performed in isolation. By specifying in the primitive assembly phase which nodes in the derivation tree form musical events, we are actually storing semantic information, since the annotation only has meaning with respect to the musical semantics phase. Other examples of this breach are the storing of note head duration and note stem direction (page 181, Figure 7.13 lines 25–26).

If strict preservation of the isolated framework is desired, such annotations could be separated from the assembly process, and moved into the musical semantics stage. This would require the introduction of a new step, executed before the graph is built, which traverses all derivation trees, annotating them accordingly. The decision to merge such annotations with the primitive assembly phase was made to simplify implementation. In



(a)



(b)

Figure 8.2: Musical events are defined by annotating nodes in the derivation tree (a) `time_atomic` (b) `secondary_atomic`.

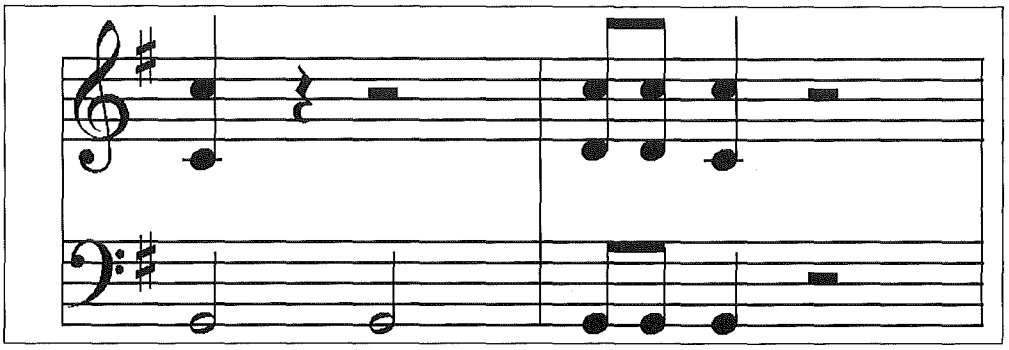


Figure 8.3: An excerpt of music before the lattice-like graph is constructed.

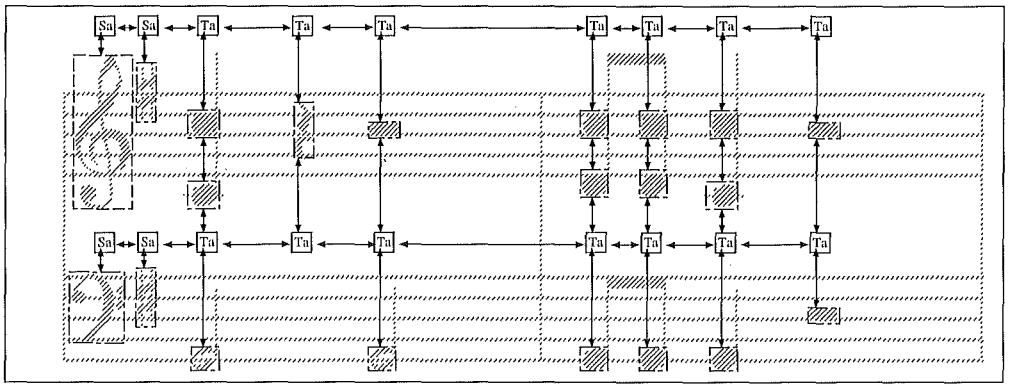


Figure 8.4: The excerpt of music after the musical events are constructed into a lattice-like graph, where links are referred to as time-threads. Ta = Time atomic, and Sa = Secondary atomic.

doing this, the only overhead is to ensure that the primitive assembly phase and the musical semantics phase agree on the attribute names used.

8.2.2 Time threads

Nodes are constructed into a lattice-like graph structure, horizontally linking events pertaining to one staff together, as well as vertically linking musical events that occur both on the same staff and adjacent staves within the same staff system. These lines are termed *time-threads.*

Figure 8.3 shows an excerpt of music and Figure 8.4 shows the corresponding lattice, with musical events linked together both horizontally and vertically. The original scanned image is shown in gray, and a dashed box has been drawn around each musical

event. Only musical events that are designated `time_atomic` are linked vertically, thus the treble clef, bass clef and two key signatures shown in the example form only horizontal links. Also, to maintain uniformity in the lattice, an extra node is created if there is no `time_atomic` event on a staff at a given point, when other `time_atomic` events exist at the same vertical alignment on other staves within the same staff system. In Figure 8.4 the node marked "Ta" beneath the crotchet rest is an example of such a node.

Every node in a derivation tree that specifies a musical event also stores a representative co-ordinate for its position on the page. It is this co-ordinate that determines the placement of the musical event within the lattice. Borrowed from cursor terminology, the *hotspot* for an event is stored using the attribute names `x_hotspot` and `y_hotspot`. Examples of this can been seen in Figure 8.2. To form horizontal threads, all events belonging to a staff (lookup `staff_system_no` and `staff_no`) are sorted based on their `x_hotspot` value. Any events that share the same value are further sorted by comparing `y_hotspot` values. To form vertical threads, only `time_atomic` events are considered, with events being sorted based on their `y_hotspot` value. Tolerances—controlled by the configurer—exist to include events in the same thread if their `x_hotspot` values differ slightly.

8.2.3 Structured lists

Horizontal time-threads are further linked so they can be traversed by *page*, *staff system*, *staff*, *bar*, or *voice*. The construct used is referred to as a *structured list*.

At the lowest level—known as the page level—all events are linked together. The start of the second staff follows on from the end of the first staff, likewise with the first and second staff systems, and so on. At other levels in the structured list, musical events are grouped by some logical notion, such as staff or staff system. *Jumper leads* exist between adjacent groups of the same type to travel from one group to another. In the case of the staff level, a jumper lead connects the first staff to the second, the second to the third, and so on. The concept *voice* refers to a level in the structured list whereby the sequence of musical events corresponding to a single part can be followed. Hence the first staff in the first staff system is linked to the first staff in the second staff system, and so on.

An example illustrating the page and bar levels of a structured list is shown in Figure 8.5. The page level never uses jumper leads. For the bar level, each time a bar line comes between two musical events in the lattice, a jumper lead is formed to bridge the gap.

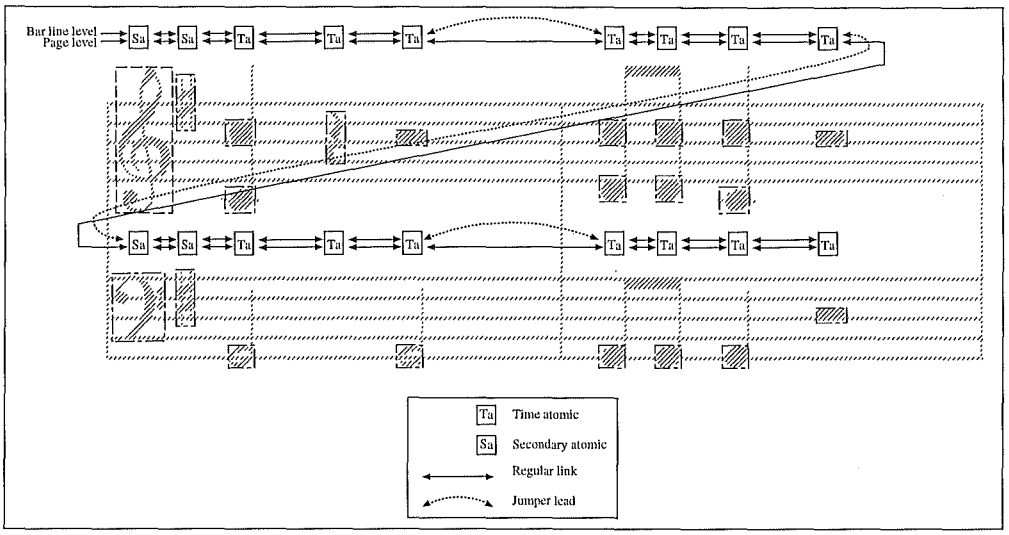


Figure 8.5: The example lattice with structured lists for bar and page built on top.

A wrap around effect occurs at the end of staves, so a bar line at the end of a staff causes a jumper lead to connect the last event on one staff to the first event on the next staff.

To build the graph it is assumed that information relating to the staff systems, staves, and bar lines is available. The first two items result from the staff detection stage of the OMR system. For bar line information, we utilise the annotation mechanism used to communicate between the primitive assembly stage and the musical semantics stage. All musical features include the attribute `mf_group` in the root node of their derivation tree, specifying the musical feature group they belong to: clef, key signature, ornament, and so on. During construction of the graph, musical features marked as `bar` for this attribute are sought and used to form jumper leads for the bar level of the structured list, based on their values of `staff_system_no`, `staff_no`, `x_hotspot`, and `y_hotspot`.

8.2.4 Applying musical semantics

Once the graph has been built, the configurer supplied routines are called to complete the musical semantics, traversing and modifying the graph. To allow for extensibility and customisation, the object oriented language C++ was selected as a vehicle for implementation. Minimalist objects for graph construction are provided by the system. Using the *inheritance* mechanism provided by the object oriented language, the configurer can add extra attributes to form their own musical event objects containing whatever information

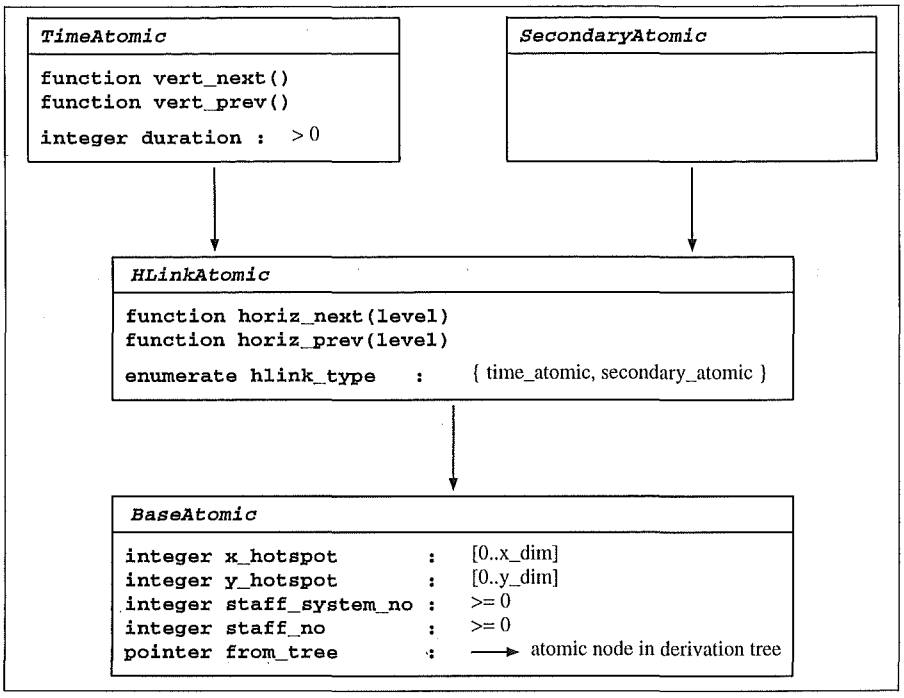


Figure 8.6: The system provides a minimalistic set of objects for graph construction.

they see fit. Inheritance is also used within the minimal set of objects provided by the system to simplify routines written by the configurer.

In Figure 8.6 the minimalistic configuration of musical events is shown. The most basic object is `BaseAtomic`. This object stores all the attributes a musical event *must* have. The field `from_tree` provides a link back to the derivation tree, so attribute names specific to a particular event can be accessed, such as `time_event_dur` in the case of a note head.

Additional objects provided by the system form an *inheritance* hierarchy. In Figure 8.6 the object `HLinkAtomic` inherits from `BaseAtomic` and can therefore access all the fields in `BaseAtomic`, such as `x_hotspot` and `from_tree`.[5] The hierarchy abstracts away detail for the convenience of the configurer. For example, when traversing a horizontal thread, the abstraction `HLinkAtomic` means that traversal code need only process objects of this type, and not worry about objects being `TimeAtomic` or `SecondaryAtomic`. The functions `horiz_next(level)` and `horiz_prev(level)` move forwards and backwards respectively, where `level` is an enumerated type specifying the level in the structured list

[5]Here we are assuming a simple form of inheritance, where an object field cannot be shielded from inherited objects.

	Order of graph transformations
1	Calculate basic pitches
2	Calculate basic durations
3	Store stem directions
4	Store beam scopes
5	Apply clefs
6	Apply key signatures
7	Apply accidentals
8	Apply time signatures
9	Apply slurs and ties
10	Apply dynamics
11	Synchronise bars

Table 8.1: The order of graph transformations.

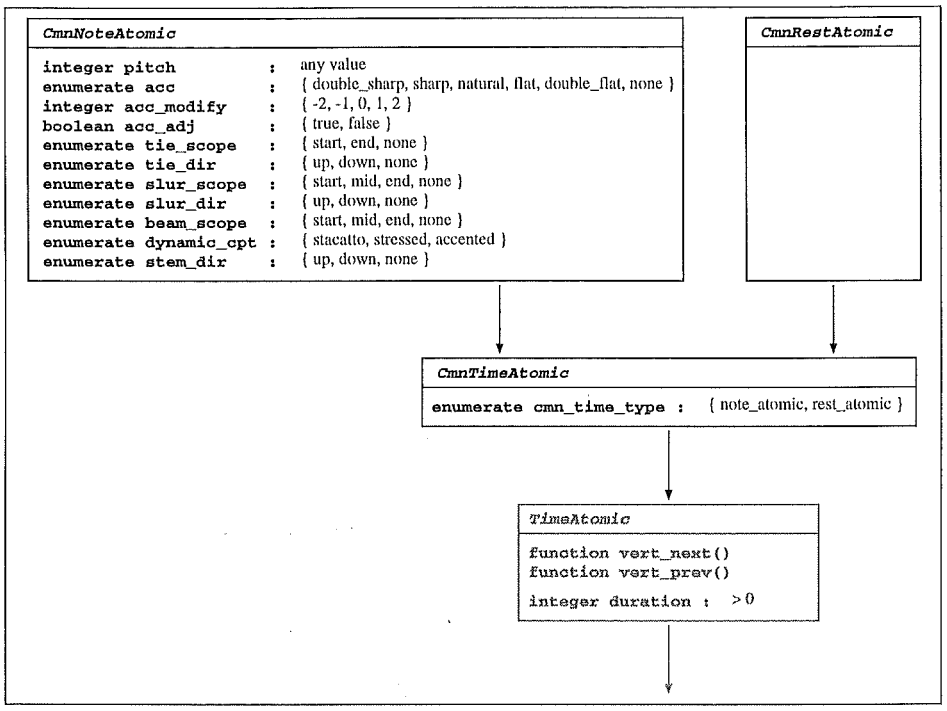
to use. When an object is found that requires modification, its type can be checked using `hlink_type` and then type cast to *`TimeAtomic`* or *`SecondaryAtomic`* appropriately.

The final layer of musical event objects in the inheritance hierarchy is defined by the configurer. Figure 8.7 details one possible setup for CMN. Events vary considerably in their needs. In this example a new object type has been created for each group of musical features classified. In Figure 8.7a *`TimeAtomic`* is subdivided into note and rest, and a representative collection of new objects for *`SecondaryAtomic`* is shown in Figure 8.7b. The roles of *`CmnTimeAtomic`* and *`CmnSecondaryAtomic`* are similar to *`HLinkAtomic`* in the minimalistic system hierarchy, serving as common objects that can be used when differentiation between the more specific object types is not required.

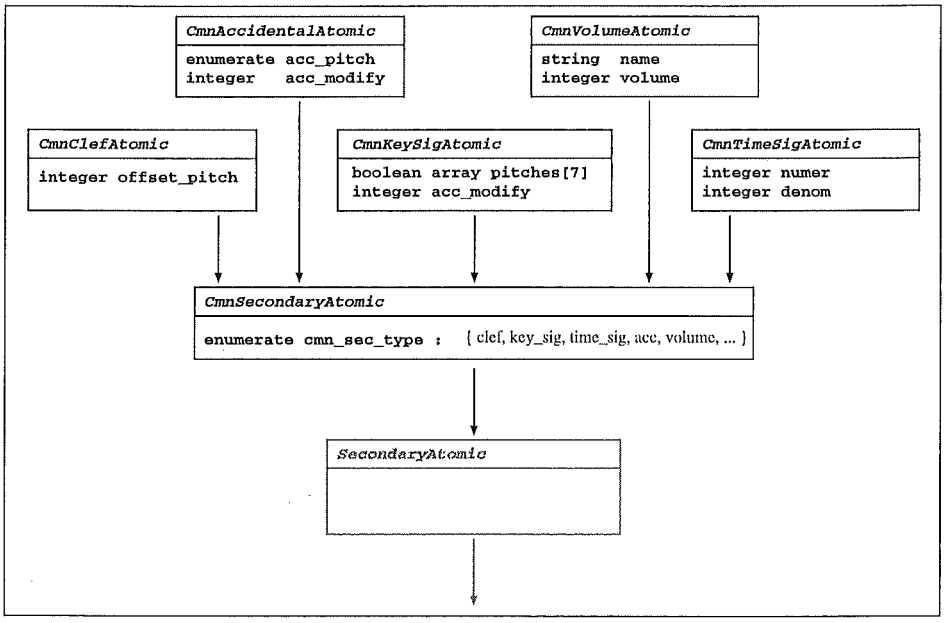
The information stored in each new object type depends on the musical features the system is configured to classify, as well as the intended application of the data, once the page has been processed semantically. In this example, a substantial set of music notation is catered for, and the information stored is intended for complete graphical reconstruction. This is why information concerning the scope and direction of graphical shapes such as a beam, a slur and a note stem is stored in *`CmnNoteAtomic`*.

Field names attempt to capture the meaning of the data stored, but for completeness we now elaborate upon the field names used in this example. This is best done by studying the steps that transform the pre-constructed graph into its final form (Table 8.1).

First, the pitch of each note head (*`CmnNoteAtomic`*) is calculated based on the staff position it occupies, and this is stored in `pitch`. Earlier in this chapter, this value was referred to as "basic pitch" (page 197). We will use the same numbering convention as be-



(a)



(b)

Figure 8.7: The configurer defines the final layer of musical events in the object hierarchy, through inheritance (a) `TimeAtomic` (b) `SecondaryAtomic`.

fore: '1' for the bottom staff line, '2' for the first staff gap above this, and so on.

Next, the duration of each note or rest (*CmnTimeAtomic*) is extracted from its derivation tree (`time_event_dur`) and stored in `duration`. This value was calculated during the primitive assembly stage and takes into account the number of tails or beams attached to the note head's stem, the type of the note head, and the number of durational dots present, but it excludes the influence of any time signature. For example, a crotchet note in 3/4 time represents one beat, but in 6/8 time the same musical feature lasts for two-thirds of a beats; however, at this point in the system, no time signature information has been taken into account, and both scenarios store the same duration. Earlier in the chapter, this was referred to as a "basic duration."

The next two steps (3 and 4) record the direction of note stems and the scope of beams, with respect to individual note heads. This information was stored during the primitive assembly stage, and therefore is a straightforward extraction process. This done, we are now in a position to apply the effects of secondary events on the primary (time) events.

The clef group (*CmnClefAtomic*) is first. For every clef in the lattice, `offset_pitch` is retrieved from its derivation tree and stored. Then the lattice is traversed at the *staff* level, modifying the pitch of all note events encountered by `offset_pitch`. This process proceeds until the end of the line is reached, or another clef is encountered, in which case the new value of `offset_pitch` supersedes the value being used. As an example, let us assume integers represent pitch, with '0' for middle C, '1' for D, '2' for E, '8' for C one octave above middle C, and so on. The offset value required for a treble clef to modify basic pitches, therefore, is '+1' and is encoded in the production rule for the treble clef in the BCG. A note head positioned over the bottom line of a staff influenced by a treble clef, therefore, has the basic pitch of '1' modified by the "apply clef" routine, resulting in the value '2,' which is indeed the first E above middle C.

A similar pattern of retrieval and application occurs for key signatures, accidentals, and time signatures, where once again precedence is given to new occurrences of the influencing object type. Key signatures traverse the lattice at the *staff* level, accidentals at the *bar* level, and time signatures at the *voice* level.

In applying the effect of a key signature, the routine accesses the boolean array `pitches` which represents the "lettered" names of the notes { C, D, E, F, G, A, B } and

indicates which accidentals are present in that particular key. The field `acc_modify` stores the modifying effect the key signature has on a note head whose pitch (modulo 8) matches an entry in the array.

By comparison, the use of data fields in the application of accidentals and time signatures is simpler. For an accidental (*CmnAccidentalAtomic*) `acc_pitch` defines the pitch of the note heads affected, and `acc_modify` defines the change in pitch required. For a time signature (*CmnTimeSigAtomic*) the value of the denominator, `denom`, is distributed to all notes and rests.

An alteration of a note's pitch due to an accidental or a key signature is represented in *CmnNoteAtomic* by an enumerated type field (`acc`) *and* a modifying integer field (`acc_modify`). This redundancy is included to simplify later routines, where sometimes it is more convenient to use the modifying pitch, and at other times the type of accidental. In addition to these fields, `acc_adj` indicates whether or not the accidental causing the alteration in pitch should be explicitly shown in the reconstructed score.

Applying the effect of slurs and ties to notes is complex since these objects can be broken over lines. Problems are further compounded by the fact that the two musical features are at times identical in graphical appearance, and can only be distinguished after consideration of context. Both these problems appear in Figure 8.8. Three ties and one slur are broken between the end of the first line and the start of the second, yet the slur starts and stops at the same height giving it the same graphical appearance as a tie. Slurs and ties can be successfully processed together by first determining the scope of each object using the *voice* level, then identifying which note heads fall within this scope. The two types of musical feature are distinguished by studying the number and pitch of the identified notes, and thus the appropriate data can be stored in the relevant *CmnNoteAtomic* objects. In the case of the object being a tie, one more step is required. If the starting note of the tie is affected by an accidental (`acc`) then this value must be propagated through to the second note which could be in another bar.

Dynamics—like slurs and ties—are free floating objects and are consequently processed in a similar manner. First the scope of a particular volume marking is determined using the *voice* level, then affected objects are modified accordingly.

Synchronisation of bars is the final step to processing the graph. Unlike previous steps, which modify the contents of nodes in the lattice, this step modifies the lattice struc-

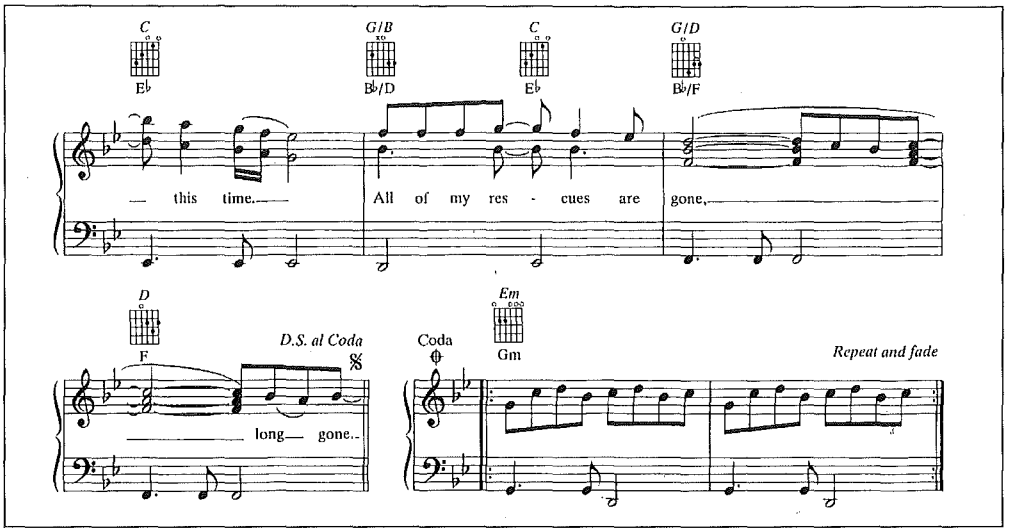


Figure 8.8: Slurs and ties are similar in appearance, and can both be broken over lines.

ture itself in an attempt to be tolerant of timing errors. For each concurrent bar within a staff system, the maximum bar duration is found, and all other bars are padded out to this length by inserting synchronisation rests (`CmnRestAtomic`) into the lattice structure at the end of each bar. Any mistakes involving timing within a bar are thus compensated for by the start of the next bar. Synchronisation is a simplistic form of error correction. A better strategy would be to use the time signature information per bar to correct any anomalies.

8.3 Evaluation

To demonstrate the flexibility of this design, the system was configured to process CMN and plainsong notation. A system almost identical to the example presented in Figure 8.7 was implemented for CMN—the only omission was dynamics (volume). The system can be extended to include other musical groups by copying the style and pattern of programming for existing groups. The configuration for plainsong notation is only for demonstration purposes, and therefore only implements a subset of the notation. For both versions, traversal routines were written to generate the musical file formats Midi, CSound, and Tilia. These formats were chosen to demonstrate the ease with which a diverse range of file formats can be accommodated: MIDI is a sound-oriented file format; Tilia is used by the music notation editor Lime; and CSound falls part-way in-between.

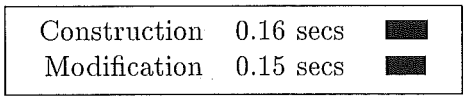


Figure 8.9: The average time taken to construct and modify the lattice-like graph calculated from processing five A4 pages of music.

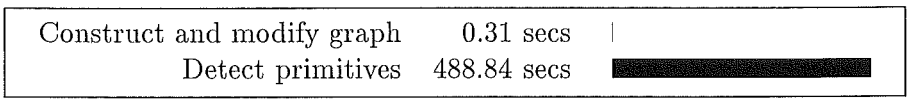


Figure 8.10: The average time taken to construct and modify the lattice-like graph compared with the average time taken to detect primitives in the same five A4 pages of music.

In evaluating the success of this stage of the OMR system, two important factors are *speed* and *accuracy*. In the results presented below we shall see that the construction and programmable modification of the graph are light computational tasks, especially when compared to other stages in the OMR system, and therefore do not merit further attention to improve efficiency. Measuring accuracy, however, is more problematic because there is no established way to calculate this. Below, we detail an accuracy metric based on the number of editing operations required to correct the computer generated piece, followed by some sample results using this metric. Tabulation of the accuracy rates from processing a larger collection of works is deferred until the next chapter (Chapter 9), where we study the entire OMR process.

8.3.1 Speed

To evaluate the computational cost for the construction and programmable modification of the lattice-like graph, five A4 pages of music were processed: the first page of "Promenade," the first two pages from "Bonita," and the first two pages from "Big My Secret." Figure 8.9 shows the average time taken to complete each step.

Both steps are extremely fast compared with other stages in the OMR system. Figure 8.10 compares the time taken to build and modify the graph with the time taken to detect primitives. Clearly there is no contest, with the musical semantics stage taking less than a second to complete its task, whereas the primitive detection stage takes over 8 minutes.

8.3.2 Accuracy

Comparing the accuracy of rival OMR systems has always been difficult. Different systems place different restrictions on the type of music that can be processed, with incompatible sets of musical features the most common difference. Even if two OMR systems process comparable collections of music notation, the numerous end-points to OMR (audio playback, graphical reconstruction, and others that lie in-between) make many comparisons ineffectual. And should two systems have compatible end-points, further complications arise due to the multitude of file formats available for the final representation. The accuracy calculation detailed below aims to minimise these problems.

First, to combat the problem of OMR systems imposing restrictions on the type of music notation processed, we utilise the extensible nature of the work described in this thesis, which naturally counteracts such limits. Second, to cope with the numerous end-points of OMR, we focus on the automatic reconstruction of a scanned image in a music notation editor as our end application. The main function of a music notation editor is to graphically display the work, but most also include a playback facility. Thus, reconstruction in a music editor covers the extreme end-points to OMR listed above. Finally, to deal with the choice of music notation editors we invent a hypothetical editor and count the number of operations it would need to correct a work, rather than becoming encumbered with the particular operations necessary in a particular editor.

The idea of calculating accuracy based on editor operations was first suggested by Carter [BC96], but as he points out, this would commit the measurement to one particular software package, warts and all. Here we expand upon the idea, with the move to a hypothetical editor.

No rigorous definition exists for the operations of the hypothetical editor, rather, if we can justify a straightforward editing operation which requires only a few keyboard[6] presses and/or clicks of the mouse, then we assign a cost of '1' to the operation. More complicated operations must be built up from these atomic steps. Some examples of simple operations are:

- adding a note,
- "cutting" a group of musical features,

[6]Either computer keyboard or musical keyboard.

- "pasting" a group of musical features,
- altering the pitch of a note, and
- changing the key signature on a staff which then automatically updates any affected notes.

When there is a series of triplets in a piece of music, a common habit in CMN is to drop the ornamental '3' after the first few occurrences. Unless the OMR system employs some form of global analysis of the detected timing information, these omissions result in mistakes that carry through to the reconstructed score. In the music editing package Finale, the conversion of three quavers beamed together into a triplet is convoluted, requiring many operations. By comparison, in the hypothetical editor we could imagine grouping the notes in question using the mouse, and then selecting a "make triplet" option from the user interface. Such a correction requires only a few keyboard and mouse presses and therefore carries the operational cost of one.

This example highlights the difference between the functionality provided by existing music notation software, and the functionality required by an OMR package. Existing editors have not been specifically designed for OMR applications, which is something that will need to change in the future. Grande Software boast the first commercial system with integrated editor and OMR software [SF94], but their description of the product talks about the OMR package being "interfaced" to the notation program rather than being developed in parallel. What the hypothetical editor allows us to assume is a working environment where the OMR system and the editor are fully integrated. The editor can always be updated to incorporate new functionality as dictated by repetitive OMR processing errors. As we shall see in Chapter 9, this theme of integration can be taken further to include the editing of data at various points in the OMR process. Such a trend can be seen emerging in commercial packages. MidiScan,[7] for instance, includes two interactive phases: the first allows the user to correct any mistakes made during staff identification, and the second allows the user to correct the musical information extracted from the page.

Figure 8.11 shows an excerpt from "Passacaglia" by J.S. Bach for the organ. Figure 8.12 shows the work reconstructed *by hand* using the music notation editor Lime.

[7]MidiScan is aimed at audio output only. For information about MidiScan, write to Musitek, 410 Bryant Circle, Suite K Ojai, CA 93023, USA.

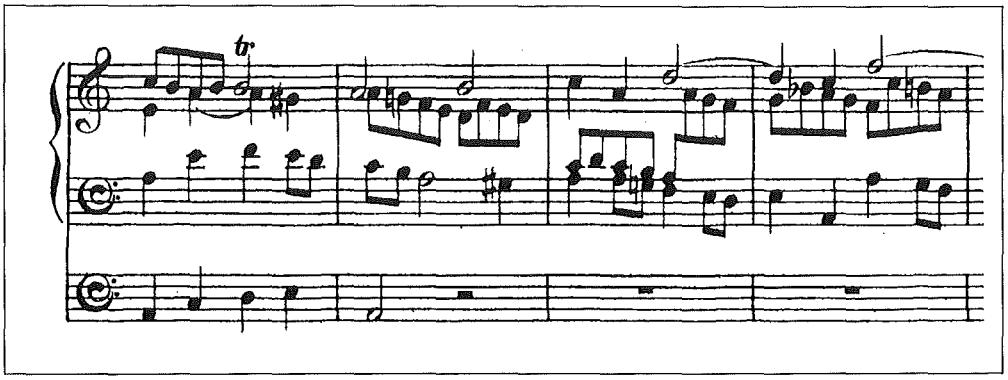


Figure 8.11: An excerpt from "Passacaglia" by J.S. Bach..

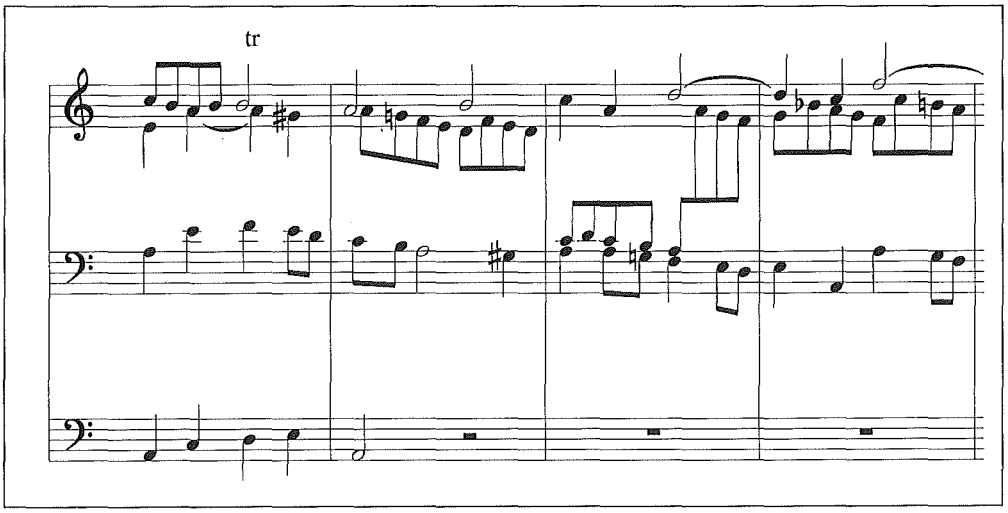


Figure 8.12: The excerpt from "Passacaglia" reconstructed by hand using the music notation editor Lime.

Without being pedantic and insisting on identical staff heights, spacing between staves, thickness of ties, and so on, the second figure is a perfect reconstruction of the music represented in the first. The proposed accuracy calculation for an OMR system is therefore calculated by the number of hypothetical operations it takes to transform the automatically reconstructed score into the "perfect" version.

The example excerpt represents an intricate piece of notation. Not only does the work include split voices, but the left-hand in bar three crosses to the staff for the right hand, and an example of one note head sharing two stems appears in bar one. Figure 8.13 shows the score reconstructed after error free primitive detection and assembly. Table 8.2

Editing operation	Cost
Add natural in bar 2	1
Convert 3 slurs to ties	3
Total	4

Table 8.2: The hypothetical operations necessary to obtain a perfect score.

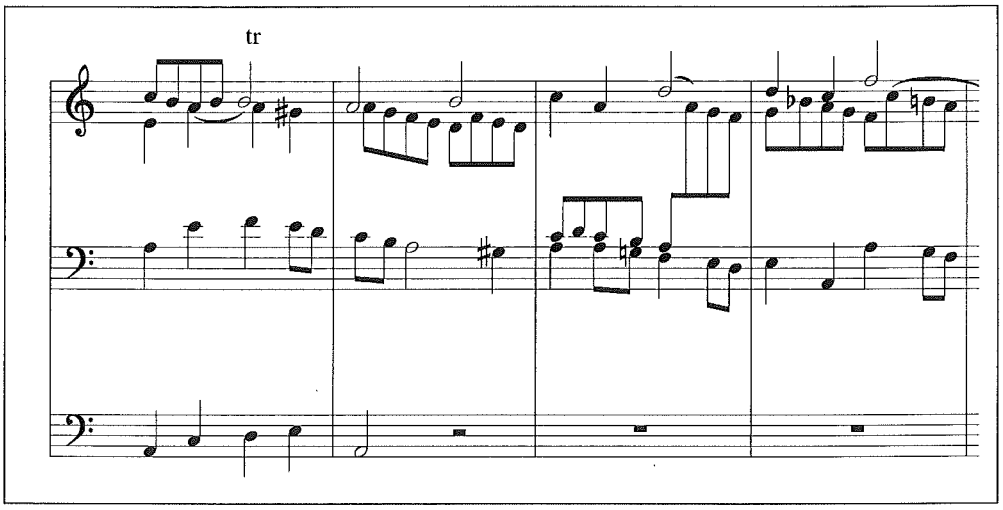


Figure 8.13: The computer generated reconstruction of "Passacaglia" in the Lime music notation editor.

lists the corrections necessary to obtain the perfect score. Like primitive assembly, the high level of structural information in musical semantics leads to high accuracy rates. The main areas of vulnerability are the calculation of pitch and determining the scope of ties and slurs. In this example, the pitch of the first natural on the first staff is incorrectly calculated as an A, and therefore does not appear in front of the note G. Also, all three ties are incorrectly processed as slurs. The hypothetical cost for complete correction, therefore, is four.

Figure 8.13 was generated in ideal circumstances, with all errors made during primitive detection and primitive assembly corrected by hand, before the musical semantics stage was invoked. This was done to illustrate the high accuracy rate inherent in this part of the system. As we have seen in Chapters 6 and 7, primitive assembly is more or less error free (it is completely error free in this example), but primitive detection is prone to making mistakes. Figure 8.14 shows the score reconstructed once more, only this time there has been no correction of errors made during primitive detection. The additional

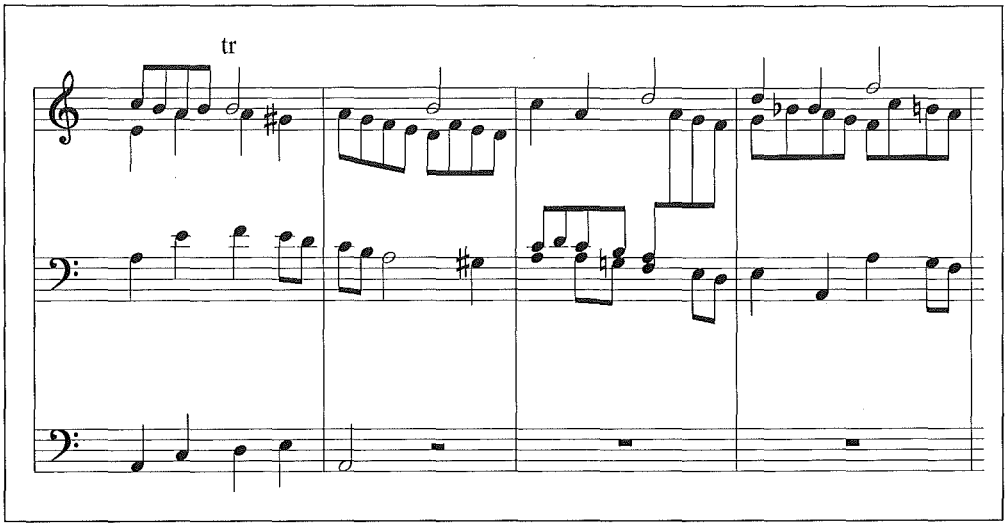


Figure 8.14: The computer generated reconstruction of "Passacaglia" in the Lime music notation editor when no corrections have been made after primitive detection.

Editing operation	Cost
Add natural in bar 2	1
Add three ties	3
Add missing minim note A (staff 1, bar 2)	1
Alter pitch of second B♭ note (staff 1, bar 4)	1
Change quaver note F to crotchet note (staff 2, bar 3)	1
Total	8

Table 8.3: The hypothetical operations necessary to obtain a perfect score when no corrections have been made after primitive detection.

errors that result are all due to failures in this stage. Table 8.3 lists the corrections necessary to generate the perfect score. As we would expect, the number of operations is higher, although for such a short excerpt it is inadvisable to draw any conclusion about the extra cost errors in primitive detection bring to the editing phase. If nothing else, this depends on the quality of notation in the starting image.

Spotting errors in the reconstructed score is time consuming because often the differences are slight. The operator must methodically work through the piece repeatedly glancing between the original and the reconstruction, looking for mistakes. In MidiScan this task is simplified by displaying a split screen, with the original image shown in the top half, and the corresponding part of the processed music shown in the bottom half.

A fully integrated editor and OMR system would extend this idea further, and in-

clude the correction of other points in the OMR process. For instance, the combined OMR system and hypothetical editor could present two images for correction after each primitive type is detected. The first image would show all the shapes detected as the current primitive type, making erroneous primitives easy to spot and remove. The second image would show the converse, i.e. the shapes not classified as the current primitive type. This is the ideal image to locate any missed primitives. Using this correction facility primitive detection errors, such as a hollow note head incorrectly classified as a filled-in note head, would be easy to spot and correct.

Similar to the accuracy measurement for score reconstruction, the accuracy of primitive detection can be expressed in terms of editing operations. Again, a hypothetical environment is assumed, where a few straightforward keyboard presses and mouse clicks form an atomic correction. To correct the wrongly classified hollow note head in the above example, the mouse pointer could be moved over the erroneous shape and the mouse button clicked. This would bring up an itemised list of possible primitives, with the marker currently showing the shape to be a filled-in note head. The mouse pointer would then be used to change the marker to denote a hollow note head, and the "OK" button pressed. This description meets our requirement of an atomic operation.

8.3.3 Plainsong notation

To demonstrate the flexibility of the described musical semantics stage, routines were developed to process a subset of plainsong notation, with the configuration targeting the musical shapes located on the staff. The main omission is the processing of lyrical text. Using our long running example for plainsong notation from the previous two chapters, Figure 8.15 shows the original image, and Figure 8.16 shows the work automatically reconstructed in Lime using CMN.

For a description of plainsong notation see "A History of Western Music" [Gro60]. As a brief explanation, a staff of plainsong notation starts with a clef symbol, followed by a sequence of notes. Both square and rhombus shapes represent notes—known as *neumes*—and (for the purposes of CMN) are taken to be of the same duration. Neumes can be built into larger shapes, forming composite neumes, with notes being played from left to right; however, a composite shape cannot exceed its associated syllable in the lyrics below. A dot after a neume doubles its value, and a horizontal line placed above or below a

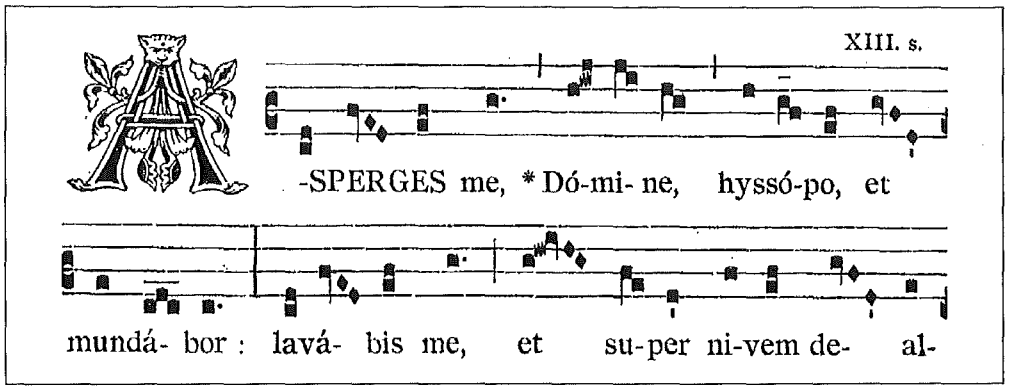


Figure 8.15: The example excerpt of plainsong notation previously used to demonstrate primitive detection and primitive assembly.

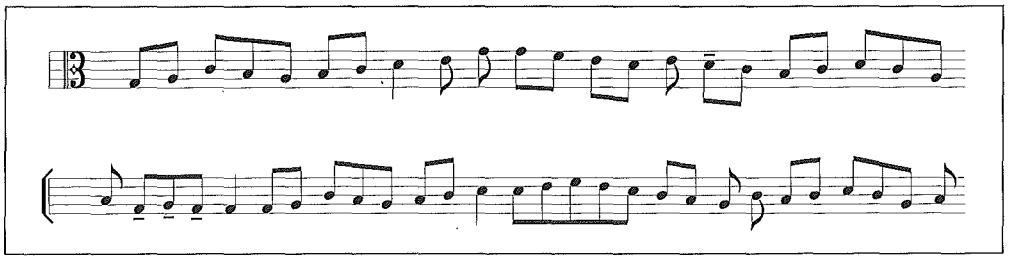


Figure 8.16: The example excerpt of plainsong notation reconstructed in Lime using CMN.

neume lengthens its duration slightly. When neumes are stacked vertically on a staff they are still played sequentially, bottom to top. Additionally, a saw-toothed shape indicates an ornamentation similar to a short trill. In the CMN reconstructed score (Figure 8.16) stress marks ('–') have been used to indicate notes in the original that were notated with slightly longer durations.

Table 8.4 lists the corrections necessary to eliminate imperfections in the reconstruction. Figure 8.17 shows the final result. Because no PRIMELA description was written for the saw-toothed shaped notation for a trilled note, no primitive is detected for the first occurrence of this shape, and the second occurrence is mistaken for a rectangle. All the editing operations required to correct the reconstructed score are a consequence of these two errors made in the primitive detection stage.

With a scanned image fully processed by the OMR system, the music can be manipulated in many ways. For instance, Figure 8.18 shows the sample excerpt of plainsong notation redisplayed using tablature notation for the guitar.

Editing operation	Cost
Add missing quaver (staff 1, above "Dó")	1
Add trill ornament (staff 1, above "Dó")	1
Beam together 3 individual quavers (staff 1, above "Dó")	1
Add trill ornament (staff 2, above "et")	1
Total	4

Table 8.4: The hypothetical operations necessary to obtain a perfect CMN version of the plainsong notation example.

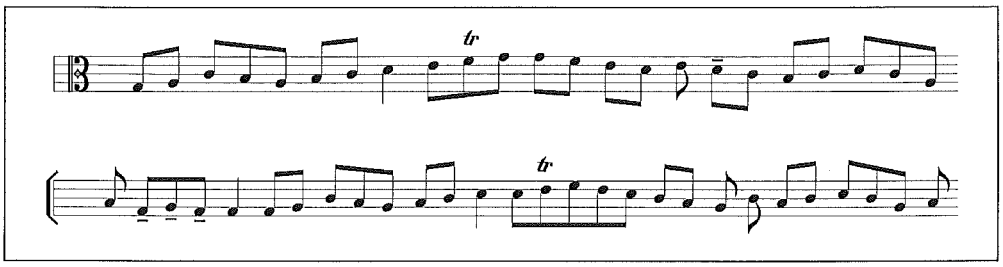


Figure 8.17: The reconstructed score of plainsong notation after the correction of imperfections.

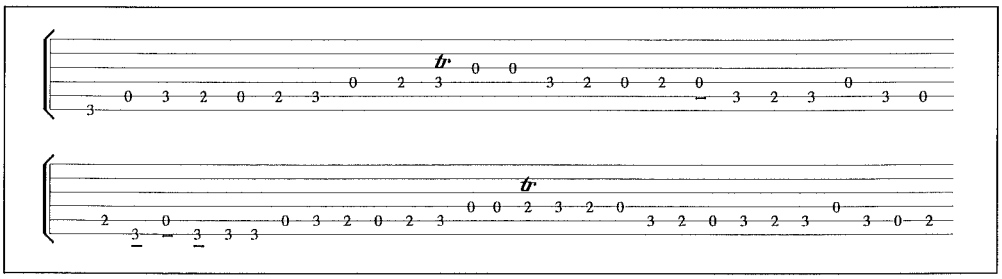


Figure 8.18: The example excerpt of plainsong notation redisplayed using tablature notation for the guitar.

8.3.4 Generating musical file formats

To indicate the versatility of the lattice-like graph structure, routines were written to generate the musical file formats Midi, CSound, and Tilia. These formats were chosen to demonstrate the ease with which a diverse range of file formats can be accommodated. Table 8.5 shows the code size for the respective formats. All routines are of a modest size. Both Tilia and Midi are binary formats requiring additional code to convert the data stored in the graph into the appropriate binary format. In contrast, CSound is a simple text format making it possible to print out the required data "on the fly." The routines

Musical file format	Traversal code	Data converter
Tilia	330	1104
CSound	185	0
Midi	263	219

(a)

Musical file format	Traversal code	Data converter
Tilia	162	1104
CSound	168	0
Midi	143	219

(b)

Table 8.5: Number of lines of code for musical file format generation (a) CMN (b) plainsong notation.

for plainsong notation are simpler then their CMN counterparts because they do not have to deal with complications like split voices, and are therefore smaller.

Chapter 9

A complete OMR system

In earlier chapters we studied the main stages to an OMR system in isolation. In this chapter we fit the pieces together to form a complete system, and study the OMR process as a whole.

By processing sample music written in CMN, tablature, and plainsong notation, Chapters 3 to 8 have demonstrated an extensible and flexible approach to OMR. In this chapter, we restrict our study of the whole OMR process to CMN—the most sought after OMR application. CMN includes a diverse range of symbols that our approach is capable of processing; however, to allow comparisons between the different categories of CMN taken from the corpus, only the core notation will be processed: clefs, key signatures, time signatures, notes, rests, accidentals, slurs, and ties.

The survey of published primitive detection work in Chapter 6 emphasises the diverse range of devised strategies. With respect to the various end applications of OMR, each strategy has its own strengths and weaknesses. Because PRIMELA does not impose any particular approach to OMR, we are able to choose a strategy that suits our needs. The primitive detection strategy chosen for this chapter is concerned with robustness and reliability, at the expense of increased computational work, and the end-application is score reconstruction.

In the past it has been claimed that an OMR system based on a single shearing operation to correct skew is faster than a rotation-based system. Although Chapter 5 confirms this to be true for the actual shearing operation, the claim makes no allowance for the time spent correcting errors that result in later stages of the system, due to skew re-

Category	Representative piece
Score	"The Cuckoo"
Miniature score	"La Primavera"
Accompaniment	"Bonita"
Monolinear (no chords)	"Music Reading: Exercise 1"
Hymn	"Hymns: Ancient & Modern"
Percussion	"Music Reading: Exercise 5"
Individual instrument (organ)	"Prelude and Fugue in G Major"
Individual instrument (guitar)	"Introduction et Variations: Sur un Motif de Rossini"
Individual instrument (piano)	"Big My Secret"
Individual instrument (popular piano)	"Hazard"
Computer typeset	"Duetto uno a Violino e Viola"

Table 9.1: The title of the representative piece from each category of printed CMN.

maining in the y-axis. This chapter investigates the claim and shows that although a robust OMR configuration does indeed cope with the remaining skew, the time saved using a single shearing operation is minimal when compared with the total time spent processing.

First we present results from an OMR configuration based on rotation that completely corrects the skew introduced through scanning. Then we present results from a configuration where skew has only been corrected in the x-axis using a single shearing operation, and compare this with the rotation-based system. For each configuration, the results tabulate the time taken to complete the various stages, and using the hypothetical editing operation count detailed in Chapter 8 (page 211) an accuracy measurement is given for the reconstructed scores.

9.1 An OMR system based on rotation

To tabulate the processing time and accuracy of a complete OMR system based on rotation, the printed CMN examples from the representational cross-section of the corpus—listed in Table 9.1—were processed.

The OMR system used was constructed from the algorithms empirically found to be the best in earlier chapters: staff detection was accomplished using the new approach detailed in Chapter 3; musical object location was accomplished using the wobble version of the staff line removal method in conjunction with the template devised by Clarke *et al.* (Chapter 4); correction for skew was accomplished by rotating the located objects (Chapter 5); primitive detection was accomplished using PRIMELA descriptions (Chapter 6); primitive assembly was accomplished using a bagged clause grammar (BCG) fil-

Primitive	Pattern recognition routines.
Treble clef	y-projection, and slices.
Bass clef curl	Hough transform, slices, and black pixel count check.
Alto clef curl	Hough transform, and black pixel count check.
Time signature	template match.
Accidental	template match, and slices.
Note tail	Hough transform, and slices.
Vertical line	Hough transform.
Beam	connectivity analysis, and feature checks.
Filled-in note head	template match.
Hollow note head	Hough transform, and black pixel count check.
Semibreve note head	template check.
Rectangular rest	template check.
Crotchet rest	y-projection.
Quaver rest	Hough transform.
Semiquaver rest	Hough transform.
Dot	template match.
Slur	connectivity analysis, and feature checks.

Table 9.2: Summary of pattern recognition routines used to classify the core primitives to CMN.

tered by staff system and embedded productions (Chapter 7); and musical semantics was accomplished using the pre-constructed lattice-like graph detailed in Chapter 8, with configurer supplied traversal routines.

Table 9.2 summarises the pattern recognition techniques used to classify the core primitives to CMN. To increase matching reliability, the implemented strategy avoids using a bounding box pre-condition if the primitive shape is likely to become fragmented or merged with adjacent shapes, and frequently more than one feature check is made to confirm classification. For example, an unknown shape is only classified as a sharp if it passes a template match, *and* two horizontal slices (one taken near the top of the shape and the other taken near the bottom) confirm the existence of two vertical lines. Additionally, when matching is restricted to areas defined by the isolated objects, dimensions are extended to compensate for noise, and uncertainty calculations are used to bolster reliability in the assembly process.

9.1.1 Processing time

Figure 9.1 shows the average time taken per image to complete each stage of the OMR system when processing the representational cross-section of printed CMN. In Figure 9.2 the most time consuming stage—primitive detection—is decomposed further, showing the

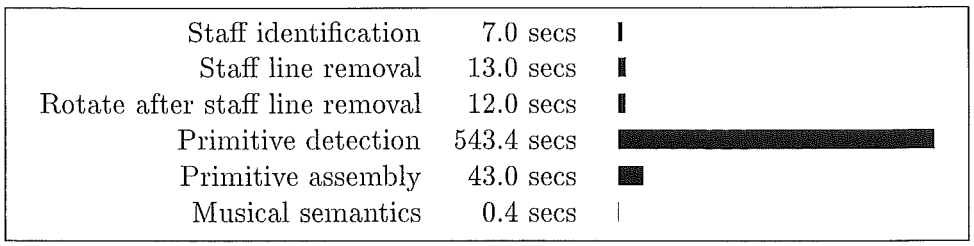


Figure 9.1: The average time taken per image to complete each stage of the OMR system based on rotation.

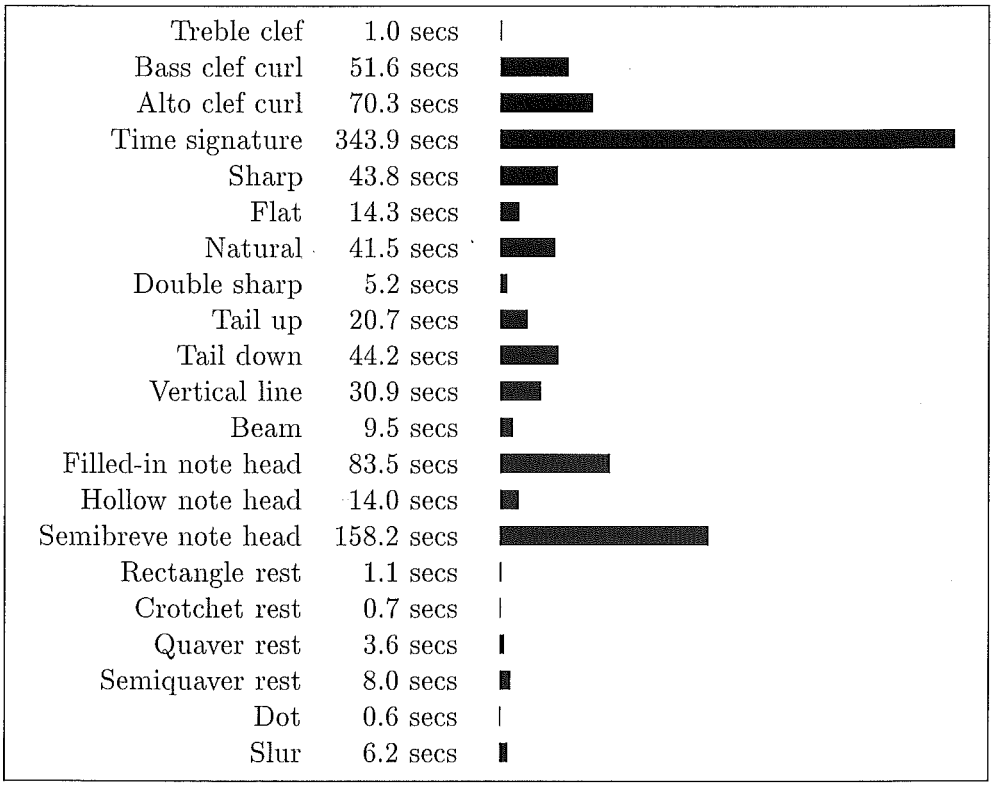


Figure 9.2: The average time taken to process the core CMN primitives for the 11 test images using an OMR system based on rotation.

time taken to process individual primitives. Despite using the multiple partitioned template optimisation proposed by Reed [Ree95], template-based PRIMELA descriptions remain the most expensive pattern recognition technique, with the detection of time signatures and semibreve note heads accounting for the majority of the time taken.

9.1.2 Accuracy

To analyse the accuracy of the system, we individually tabulate the results from the four principal OMR stages: staff detection, primitive detection, primitive assembly, and musical semantics.

Staff detection

For the 11 test images, the OMR system correctly identifies all staff systems, staves, and staff lines. Although we cannot conclude that the developed staff detection algorithm will be guaranteed perfect all the time, the sample selection indicates an extremely high level of accuracy for a wide range of works.

Primitive detection

Tables 9.3 and 9.4 present accuracy statistics for the primitive detection stage, showing the percentage of correctly classified primitives, the percentage of missed primitives, and the percentage of incorrectly classified shapes. The bottom line of each table shows the total number of primitives present, and the weighted average for each of the percentage columns.

In the normal calculation for the average, all data values are of equal importance. This is problematic when calculating an average value for primitive detection because the sample size varies for different primitive types. For example, the accuracy rate for vertical line detection is based on a sample size of 3224, whereas the accuracy rate for semibreve notes is based on a sample size of two.

To avoid low sample sizes distorting our calculation, we calculate the weighted average:

$$\text{Weighted average} = \frac{\sum_{i=0}^{N} \left(\frac{ss_i}{tss/N} \cdot \frac{rv_i}{ss_i}\right)}{N} \tag{9.1}$$

where:

N is the number of values,

rv_i is a recorded value,

ss_i is the sample size for value i, and

tss is the total number of samples ($\sum_{i=0}^{N} ss_i$).

Or equivalently:

$$\text{Weighted average} = \frac{\sum_{i=0}^{N} rv_i}{\sum_{i=0}^{N} ss_i}. \tag{9.2}$$

Table 9.3 shows that the bulk of the primitives are processed correctly, with most match rates for primitives above 95%. Notable exceptions are naturals, hollow note heads, and crotchet rests, where over 10% of each primitive type is missed.

An investigation of the omitted objects revealed a high level of deformation in naturals and hollow note heads due to staff line removal. For a natural, small parts of staff line sometimes remain attached to the object, and for a hollow note head, parts that blend into a staff line are incorrectly erased. Such deformations were anticipated for these (and other) primitives, however the robustness of the current descriptions for naturals and hollow note heads are insufficient for the task set.

By contrast, only small deformities were found on the omitted crotchet rests, but classification failure was still due to insufficient robustness. Small variations in the vertical position of crotchet rests on staves cause small staff line appendages to occur at slightly different positions on each rest. This in turn alters a rest's y-projection which, in some cases, fails to match the ideal projection.

Primitive detection accuracy rate

Calculating an overall accuracy rate for primitive detection is complicated by the use of uncertain data because some incorrect classifications do not lead to errors during primitive assembly and musical semantics. This is the reason for the "dangerously incorrect" column in Table 9.4, which records how many of the incorrectly classified shapes will cause errors in subsequent stages. Missed primitives also result in errors, and thus an overall accuracy rate for the system is:

Primitive	Number of primitives	Percentage of primitives correct (certainty≥1.0)	Percentage of primitives correct (certainty<1.0)	Percentage of primitives missed
Treble clef	67	98.51%	0.00%	1.49%
Bass clef curl	39	97.44%	0.00%	2.56%
Alto clef curl	8	100.00%	0.00%	0.00%
Time signature	23	100.00%	0.00%	0.00%
Sharp	219	93.61%	0.46%	5.94%
Flat	92	95.65%	1.09%	3.26%
Natural	57	82.46%	0.00%	17.54%
Double sharp	2	100.00%	0.00%	0.00%
Tail up	215	98.60%	0.00%	1.40%
Tail down	131	95.42%	0.00%	4.58%
Vertical line	3224	98.85%	0.00%	1.15%
Beam	791	91.53%	0.51%	7.96%
Filled-in note head	2830	98.98%	0.95%	0.07%
Hollow note head	174	81.61%	0.00%	18.39%
Semibreve note head	2	100.00%	0.00%	0.00%
Rectangle rest	119	99.16%	0.00%	0.84%
Crotchet rest	80	82.50%	6.25%	11.25%
Quaver rest	281	98.58%	0.00%	1.42%
Semiquaver rest	14	100.00%	0.00%	0.00%
Dot	334	96.41%	0.00%	3.59%
Slur	244	94.26%	0.00%	5.74%
Total/Weighted average	8946	97.22%	0.42%	2.36%

Table 9.3: Accuracy statistics for the correct classification of the core CMN primitives in the 11 test images using an OMR system based on rotation.

Primitive	Number of primitives	Percentage of primitives incorrect (certainty≥1.0)	Percentage of primitives incorrect (certainty<1.0)	Percentage of primitives dangerously incorrect
Treble clef	67	0.00%	0.00%	0.00%
Bass clef curl	39	0.00%	0.00%	0.00%
Alto clef curl	8	0.00%	0.00%	0.00%
Time signature	23	0.00%	0.00%	0.00%
Sharp	219	0.00%	0.46%	0.00%
Flat	92	2.17%	8.70%	2.17%
Natural	57	0.00%	0.00%	0.00%
Double sharp	2	0.00%	0.00%	0.00%
Tail up	215	2.33%	0.00%	1.86%
Tail down	131	0.00%	0.00%	0.00%
Vertical line	3224	0.78%	0.00%	0.43%
Beam	791	0.88%	0.38%	0.88%
Filled-in note head	2830	0.74%	0.39%	0.74%
Hollow note head	174	4.60%	0.00%	4.02%
Semibreve note head	2	0.00%	100.00%	0.00%
Rectangle rest	119	0.00%	0.00%	0.00%
Crotchet rest	80	0.00%	0.00%	0.00%
Quaver rest	281	2.14%	0.00%	2.14%
Semiquaver rest	14	0.00%	0.00%	0.00%
Dot	334	3.89%	0.00%	2.99%
Slur	244	0.82%	0.00%	0.00%
Total/Weighted average	8946	0.99%	0.28%	0.79%

Table 9.4: Accuracy statistics for the incorrect classification of the core CMN primitives in the 11 test images using an OMR system based on rotation.

Primitive	Number of primitives	Number of primitives missed	Number of primitives dangerously incorrect	Overall accuracy rate
Treble clef	67	1	0	98.51%
Bass clef curl	39	1	0	97.44%
Alto clef curl	8	0	0	100.00%
Time signature	23	0	0	100.00%
Sharp	219	13	0	94.06%
Flat	92	3	2	94.57%
Natural	57	10	0	82.46%
Double sharp	2	0	0	100.00%
Tail up	215	3	4	96.74%
Tail down	131	6	0	95.42%
Vertical line	3224	37	14	98.42%
Beam	791	63	7	91.15%
Filled-in note head	2830	2	21	99.19%
Hollow note head	174	32	7	77.59%
Semibreve note head	2	0	0	100.00%
Rectangle rest	119	1	0	99.16%
Crotchet rest	80	9	0	88.75%
Quaver rest	281	4	6	96.44%
Semiquaver rest	14	0	0	100.00%
Dot	334	12	10	93.41%
Slur	244	14	0	94.26%
Total/Weighted average	8946	211	71	96.85%

Table 9.5: Overall accuracy rate for the core CMN primitives in the 11 test images using an OMR system based on rotation.

$$\text{Primitive detection accuracy rate} = 1 - \frac{missed + dangerously\_wrong}{total\_num\_prims} \tag{9.3}$$

where:

missed is the number of primitive shapes missed,

dangerously_wrong is the number of dangerously incorrect primitives, and

total_num_prims is the number of primitives contained in the image.

Table 9.5 shows the result of applying this calculation to the tabulated data. Although OCR systems with accuracy rates of over 99% are commonplace, an average accuracy rate of 96.85% for the primitive detection stage of an OMR system indicates a reliable configuration for a more complex task. The accuracy rate could be improved through the refinement of PRIMELA descriptions. Table 9.5 highlights the weaker descriptions.

An alternative way to study the data is to compute the average accuracy rate for each sample image. This has been done in Table 9.6. The sample monolinear image, "Music Reading: Exercise 1," has the poorest accuracy rate at 94.42%, due to a high number of missed primitives. All other accuracy rates are above 96%.

Category	Average overall accuracy rate
Orchestrated score	98.58%
Miniature score	97.16%
Accompaniment	98.43%
Monolinear (no chords)	94.42%
Organ	97.96%
Guitar	97.35%
Piano	97.99%
Popular piano	96.55%
Hymn	98.79%
Percussion	98.25%
Computer generated	98.22%

Table 9.6: Overall accuracy rate for each sample image using an OMR system based on rotation.

Primitive assembly

The accuracy calculation for primitive assembly is based on the number of *assembly operations* performed, where an assembly operation is defined as the act of adding a primitive shape to an already existing derivation tree. To calculate the accuracy rate for primitive assembly:

$$\text{Primitive assembly accuracy rate} = 1 - \frac{missed\_ass + wrong\_ass}{total\_num\_ass} \tag{9.4}$$

where:

$missed\_ass$ is the number of missed assembly operations,

$wrong\_ass$ is the number of incorrect assembly operations, and

$total\_num\_ass$ is the number of primitives contained in the image.

Like the calculation for primitive detection, the formula takes into account both the omitted operations, and the operations that should not have occurred.

Table 9.7 shows the accuracy rate for each sample image, where errors made during the primitive detection stage have already been corrected. At 83.80%, the assembly rate for the miniature score example, "La Primavera," is notably lower than other images. For this image, all errors are due to missed assembly operations, with the largest type of mistake being the failure to assemble note stems with the appropriate beam. Out of a possible 156 stem to beam assembly operations, only 120 were performed.

Category	Number of assembly operations	Number of missed assembly operations	Number of incorrect assembly operations	Overall accuracy rate
Orchestrated score	554	0	0	100.00%
Miniature score	395	64	0	83.80%
Accompaniment	356	10	0	97.19%
Monolinear (no chords)	580	4	0	99.31%
Organ	551	3	0	99.46%
Guitar	685	21	1	96.79%
Piano	581	7	0	98.80%
Popular piano	303	2	0	99.34%
Hymn	289	20	5	91.35%
Percussion	97	2	0	97.94%
Computer generated	1168	1	0	99.91%
Total/Weighted average	5559	134	6	97.48%

Table 9.7: Overall primitive assembly accuracy rate for each sample image using an OMR system based on rotation.

Error type	Missed operation	Incorrect operation
Note head + stem	44	6
Tail + stem	12	0
Stem + beam	62	0
Dot + note head	16	0

Table 9.8: Statistics on assembly errors for the core CMN primitives in the 11 test images using an OMR system based on rotation.

The totals in the table for missed operations and incorrect operations (134 and 6 respectively) show a strong bias towards missed operations. Table 9.8 gives a more detailed account of the type of errors that occurred. All six incorrect assembly errors are of the same type, and all occur during passages of notation using split voice. Figure 9.3 shows one such mistake taken from the hymn example. Because there is no distinct limit to the bottom of the rightmost stem, the vertical line detected extends into the lower note head, and thus both note heads meet the proximity constraint necessary for assembly with the rightmost stem.

The wide range and high values in error type for missed operations implies that many of the tolerances that constrain the proximity of primitive shapes for valid assembly are too strict. The narrow range and low values in error type for incorrect operations implies that a loosening of many constraints will not increase incorrect assembly errors. Such a modification would greatly improve the assembly rate for the miniature score. Discounting the statistics for the miniature score, due to the over-strict assembly process, yields an overall accuracy rate of 98.53% for the 10 remaining pieces.

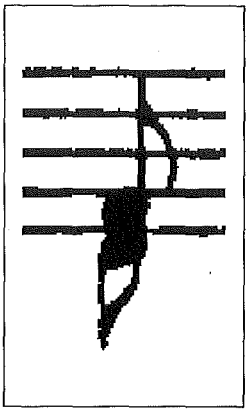


Figure 9.3: An example of split voice notation that causes the incorrect assembly of the lower note head with the rightmost stem.

Error type	Number required
Incorrect lattice structure	2
Incorrect note pitch	35
Incorrect ♭ pitch	1
Incorrect ♮ pitch	6
Incorrect ♯ pitch	6
Incorrect slur/tie scope	7
Incorrectly processed as slur	1
Incorrectly processed as tie	0

Table 9.9: Statistics on musical semantic errors for the core of CMN using an OMR system based on rotation.

Musical semantics

Using the hypothetical music notation editor described in the previous chapter (Section 8.3.2, page 211), we measure the accuracy of the musical semantics stage by counting the number of editing operations required to transform the computer generated score into the desired score. For comparison reasons we restricted the class of processed shapes to the core notation. It is only fair, therefore, to restrict the desired reconstruction to one that uses only these shapes. Table 9.9 shows the type and number of mistakes made in processing the printed CMN representational cross-section.

Predictably the number of incorrect note pitches is the most common form of mistake because they are the most common feature in the test images. The majority of these errors occur when a note head pitch is specified using ledger lines because the algorithm used to calculate pitch assumes that the lines for pitch (be it continuous staff lines or in-

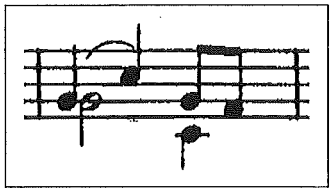


Figure 9.4: A split voice occurring at the same pitch can cause two concurrent notes to be incorrectly linked sequentially in the lattice-like graph.

termittent ledger lines) occur at a uniform vertical spacing. This is not true for ledger lines in some works, where the separation between pitches is increased. An improved pitch calculation would take into account the number of ledger lines attached to a note stem.

The "incorrect lattice structure" entry in the table records the number of mistakes that occur in the structure of the lattice-like graph. For the images tested, both errors result from sections of music written using split voices, where concurrent note heads from an upper and lower voice that should have shared a vertical time-thread, did not. Figure 9.4 illustrates the problem. The first crotchet and the first minim mark the start of a split voice section (upper and lower voice respectively), and should therefore be played at the same time. However, the distance between the centre of each note head exceeds the computer calculated threshold for note heads in the same vertical time-thread, causing the two time events to be incorrectly linked sequentially as part of a horizontal time-thread.

Relaxing the computer calculated threshold to include these events is unwise because there are other situations in music notation—such as a passage of semiquaver notes tightly packed together—where a sequential series of notes can be legitimately spaced at a comparable distance to that which separates the crotchet/minim split voice. A better solution is to analyse the wider context of musical events—for instance, a post-processing step that considers the duration of measures and, based on stem direction, restructures the lattice-like graph when inconsistencies are encountered.

Musical semantics accuracy rate

To express the accuracy of the musical semantics stage, counting the number of editing operations required to correct the score is only part of the calculation, since this value does not reflect the number of score reconstruction operations that have been correctly

performed. Just like editing a score though, there is more than one way to accomplish this task.

To simplify things, we shall use the number of musical events in a work[1] as a measure of the number of successful score reconstruction operations performed, since this value reflects the complexity of the piece. The accuracy calculation is thus:

$$\text{Musical semantics accuracy rate} = 1 - \frac{hypothetical\_edit\_ops}{total\_num\_musical\_events} \tag{9.5}$$

where:

hypothetical_edit_ops

is the number of hypothetical editing operations required to correct the score, and

total_num_musical_events

is the number of musical events contained in the image.

Table 9.10 shows the result of applying this calculation to each image in the representational cross-section of printed CMN, where errors made during the primitive assembly stage have already been corrected. Like primitive assembly, the high level of structure inherent in musical semantics yields extremely high accuracy rates. Only three pieces fall below 98% (the guitar example "Introduction et Variations: Sur un Motif de Rossini," the hymn example "Hymns: Ancient & Modern," and the percussion example "Music Reading: Exercise 5") resulting in an overall average accuracy rate of 98.57%.

A collective accuracy rate

Now that we have accuracy rates for each key stage in the OMR process, we can present a collective accuracy rate for the system. There is more than one way to accomplish this. Calculating the mean presents an average accuracy rating, but this can also mask important weaknesses. For instance, an OMR system with perfect staff detection, primitive assembly, and musical semantics, combined with a disastrous primitive detection stage would have an average accuracy rate of at least 75%. Conversely, representing the collective accuracy rate of an OMR system by the stage with the lowest value also ignores important information.

[1]Recall a musical event is defined during the primitive assembly stage by annotating a node in the derivation tree with either `time_atomic` or `secondary_atomic`. Each musical event then goes on to form a node in the lattice-like graph built by the musical semantics stage.

Category	Accuracy rate
Orchestrated score	99.22%
Miniature score	98.54%
Accompaniment	98.52%
Monolinear (no chords)	98.95%
Organ	98.71%
Guitar	96.44%
Piano	98.81%
Popular piano	99.22%
Hymn	97.20%
Percussion	96.77%
Computer generated	99.47%
Weighted average	98.57%

Table 9.10: Musical semantics accuracy rate for each sample image using an OMR system based on rotation.

Rather than summarise these four values into one we choose to express the information as the tuple:

(*staff detection*, *primitive detection*, *primitive assembly*, *musical semantics*).

Hence the collective accuracy rate of the OMR configuration based on rotation is:

(100%, 96.85%, 97.48%, 98.57%).

This information summarises the accuracy of the OMR system. If we so desire, each item in the tuple can be expanded to reveal more detailed information about the system using the data presented earlier in the chapter.

9.2 An OMR system based on shearing

To tabulate the processing time and accuracy of an OMR system configured to correct skew only in the x-axis using a shearing operation, the representative printed CMN examples were once again processed. The configuration used was identical to the rotation based system (Section 9.1) except the correction for skew stage used a shearing operation.

9.2.1 Processing time

Figure 9.5 shows the average time taken per image to complete each stage. Although the time taken to "shear after staff line removal" is much faster than the rotation-based counterpart "rotate after staff line removal" (1.42 seconds compared with 12.02 seconds), the

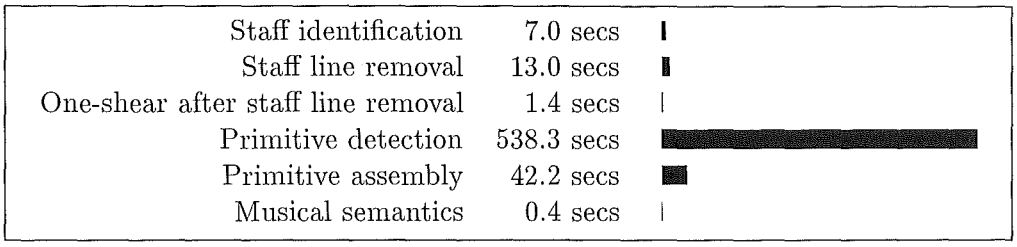


Figure 9.5: The average time taken per image to complete each stage of the OMR system based on one shearing operation.

saving is overshadowed by the cost of the primitive detection stage. The shearing-based system takes on average 602.34 seconds to process an image from start to finish, compared with 618.85 seconds for the rotation-based system. This is only a saving of 2.7%.

9.2.2 Accuracy

Accuracy ratings are expressed using the calculations developed in the previous section (Equations 9.3 to 9.5).

Staff detection

The two OMR systems under scrutiny (shearing and rotation) are identical in configuration initially. They only start to diverge at the correction for skew step. Consequently, the staff detection accuracy rate for the shearing-based system is identical to the rotation-based system, which for the 11 test images is 100%.

Primitive detection

For primitive detection we expect the accuracy results for the two systems to diverge. Moreover, with the shearing-based configuration only correcting skew in the x-axis, we expect to see an increase in the number of errors—particularly vertical line detection.

Table 9.11 compares the accuracy rates for the two configurations. A negative value in the last column indicates a better accuracy rate by the shearing-based system, and a positive value indicates a better accuracy rate by the rotation-based system. Surprisingly, the overall accuracy of the shearing-based system is *better* than the rotation-based system, though only by a fraction of a percent (0.056%).

Primitive	Accuracy rate (Rotate)	Accuracy rate (Shear)	Rotate − Shear
Treble clef	98.51%	98.51%	0.00%
Bass clef curl	97.44%	97.44%	0.00%
Alto clef curl	100.00%	100.00%	0.00%
Time signature	100.00%	100.00%	0.00%
Sharp	94.06%	94.52%	−0.46%
Flat	94.56%	94.56%	0.00%
Natural	82.46%	82.46%	0.00%
Double sharp	100.00%	100.00%	0.00%
Tail up	96.74%	96.74%	0.00%
Tail down	95.42%	95.42%	0.00%
Vertical line	98.42%	98.54%	−0.12%
Beam	91.15%	92.16%	−1.01%
Filled-in note head	99.19%	99.19%	0.00%
Hollow note head	77.59%	75.29%	+2.30%
Semibreve note head	100.00%	100.00%	0.00%
Rectangle rest	99.16%	99.16%	0.00%
Crotchet rest	88.75%	87.50%	+1.25%
Quaver rest	96.44%	96.44%	0.00%
Semiquaver rest	100.00%	100.00%	0.00%
Dot	93.41%	92.51%	+0.90%
Slur	94.26%	94.26%	0.00%
Weighted average	96.85%	96.90%	−0.05%

Table 9.11: A comparison of the accuracy rates for the core CMN primitives in the 11 test images using OMR systems based on rotation and shearing.

The fact that the accuracy of the two configurations are so close supports the claim that the implemented PRIMELA descriptions are robust and reliable. This is exemplified by the description for vertical lines, where there is a 0.124% increase in accuracy by the shearing-based system, despite the crookedness that remains in the vertical axis.

To understand this unexpected robustness, we must delve deeper into the details of the PRIMELA description for a vertical line. In Chapter 6, two PRIMELA descriptions were used to recognise all possible vertical lines: long and short. For the OMR system used to process the printed CMN representational cross-section, three vertical line descriptions were used: long, medium, and short. All three descriptions rely on the Hough transform, using a vertical line for the parametric equation. The only difference is the length of line used.

In all three descriptions, the parametric equation passed to the Hough transform describes a vertical line one pixel wide, but of course the real vertical lines have a greater thickness than this. To obtain the location of a vertical line in the image, adjacent matches of one-pixel wide lines are merged together. Finally, a post-conditional check, which tolerates moderately slanting vertical lines, is used to ensure the merged object does not become too wide.

Type	Rotate	Shear	Rotate − Shear
Long vertical line	461	450	11
Medium vertical line	572	579	-7
Short vertical line	2154	2161	-7

Table 9.12: A comparison of the number and type of vertical lines detected using OMR systems based on rotation and shearing.

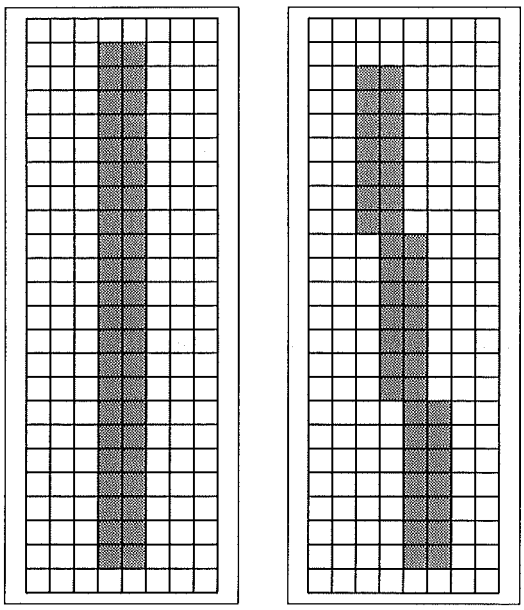
Category	Accuracy rate (Rotate)	Accuracy rate (Shear)	Rotate − Shear
Orchestrated score	98.58%	98.67%	−0.09%
Miniature score	97.16%	97.16%	0.00%
Accompaniment	98.43%	98.27%	+0.16%
Monolinear (no chords)	94.42%	93.99%	+0.43%
Organ	97.96%	97.84%	+0.12%
Guitar	97.35%	97.99%	−0.64%
Piano	97.99%	98.71%	−0.72%
Popular piano	96.55%	96.55%	0.00%
Hymn	98.79%	98.62%	+0.17%
Percussion	98.25%	98.25%	0.00%
Computer generated	98.22%	98.22%	0.00%

Table 9.13: A comparison of the primitive detection accuracy rate for each sample image using OMR systems based on rotation and shearing.

Table 9.12 decomposes the number of vertical lines detected into their three respective categories. The data shows there is a shift in the shearing-based system away from long vertical lines towards medium and short vertical lines.

The shift is due to the skew that remains in vertical lines after shearing. Figure 9.6 shows the same vertical line after correction for skew by rotation (Figure 9.6a) and by shearing (Figure 9.6b). In the first situation, the *long* vertical line description detects the primitive, but in the second situation, there are not enough black pixels in a vertical column to pass the test. Assuming the angle of skew is low, the reduced number of black pixels in a column will pass the test for either the *medium* vertical line or *short* vertical line. These one pixel wide matches are then merged together to form the original (moderately slanted) vertical line, which is then subjected to the post-conditional check.

Table 9.13 compares the average accuracy rate for each sample image. Again we see close accuracy rates for the two configurations, with the accuracy of the shearing-based system never more than 0.8% away from the equivalent rotation-based value.



(a) (b)

Figure 9.6: A vertical line after correction for skew (a) rotation (b) shearing.

Primitive assembly

Table 9.14 compares the primitive assembly accuracy rate of the two configurations for each sample image. Like primitive detection, the differences are small—never more than 0.5%—confirming that the tolerances used in the assembly rules adequately cope with the fluctuations introduced by processing images that have been corrected for skew using a single shearing operation.

Musical semantics

Table 9.15 compares the musical semantics accuracy rate of the two configurations for each sample image. Nine of the 11 images produce identical results. For the remaining two images (the accompaniment example "Bonita," and monolinear example "Music Reading: Exercise 1") the rotation-based system yields better results because the shearing-based system makes more errors in calculating the pitch of notes. The differences are shown in Table 9.16.

Category	Accuracy rate (Rotate)	Accuracy rate (Shear)	Rotate − Shear
Orchestrated score	100.00%	100.00%	0.00%
Miniature score	83.80%	83.88%	−0.08%
Accompaniment	97.19%	97.37%	−0.18%
Monolinear (no chords)	99.31%	98.97%	+0.34%
Organ	99.46%	99.26%	+0.20%
Guitar	96.79%	96.47%	+0.32%
Piano	98.80%	98.80%	0.00%
Popular piano	99.34%	99.67%	−0.33%
Hymn	91.35%	90.94%	+0.41%
Percussion	97.94%	98.10%	−0.16%
Computer generated	99.91%	99.91%	0.00%
Weighted average	97.48%	97.40%	+0.08%

Table 9.14: A comparison of the primitive assembly accuracy rate for each sample image using OMR systems based on rotation and shearing.

Category	Accuracy rate (Rotate)	Accuracy rate (Shear)	Rotate − Shear
Orchestrated score	99.22%	99.22%	0.00%
Miniature score	98.54%	98.54%	0.00%
Accompaniment	98.52%	94.38%	+4.14%
Monolinear (no chords)	98.95%	98.73%	+0.22%
Organ	98.71%	98.71%	0.00%
Guitar	96.44%	96.44%	0.00%
Piano	98.81%	98.81%	0.00%
Popular piano	99.22%	99.22%	0.00%
Hymn	97.20%	97.20%	0.00%
Percussion	96.77%	96.77%	0.00%
Computer generated	99.47%	99.47%	0.00%
Weighted average	98.57%	98.20%	+0.37%

Table 9.15: A comparison of the musical semantics accuracy rate for each sample image using OMR systems based on rotation and shearing.

A collective accuracy rate

Table 9.17 compares the collective accuracy rate of the two configurations. As we have seen in the above sections, the differences in accuracy rate that occur at each stage are small—virtually insignificant. Both configurations yield identical staff detection accuracy rates; the shearing-based system is marginally better at primitive detection; and the rotation-based system is marginally better at primitive assembly, and musical semantics. We also saw above that a shearing-based system is only slightly faster than a rotation-based system (an improvement of 2.7%). Thus, the small speed-up accomplished is countered by a small increase in the number of errors made.

Category	Miscalculated note pitches (Rotate)	Miscalculated note pitches (Shear)
Accompaniment	3	17
Monolinear (no chords)	0	1

Table 9.16: A comparison of the note pitch miscalculations made by the two OMR configurations.

Configuration	Collective accuracy rate
Rotate	(100%, 96.85%, 97.48%, 98.57%)
Shear	(100%, 96.90%, 97.40%, 98.20%)

Table 9.17: A comparison of the collective accuracy rates for the two OMR configurations.

9.2.3 Accuracy rates of existing OMR systems

In Section 8.3.2 we highlighted the difficulty in comparing the accuracy rates of OMR systems. Reed [Ree95] demonstrates the problem with an incisive example in which he presents, from the same set of data, two different calculations that yield accuracy levels of 11% and 95% respectively. We therefore make no comparison of published accuracy results with the OMR system detailed in this thesis.

For completeness, we now summarise the reported accuracy rates of prominent OMR systems. Generally, authors express the number of correctly processed shapes as a percentage of the shapes the OMR system *should* have processed: the Wabot-2 project [MHS+85] reports an accuracy rate of "nearly 100%" for ten simple scores of organ music; Kato and Inokuchi [KI90] report accuracy rates of 83–95% (an average of 89%) for four test scores of varying complexity; Modayur *et al.* [MRHS95] report an overall accuracy rate of 96% for 74 images (however these images are small—the total surface area of the 74 images is approximately equivalent to two A4 pages); and Reed [Ree95] reports accuracy rates in the range 78–100% (an average of 95%) for nine images of printed CMN.

Chapter 10

Conclusion

This thesis investigated the requirements of an extensible OMR system. Work was divided into six principal stages—staff detection, musical object location, image enhancement, primitive detection, primitive assembly, and musical semantics—with each stage studied in isolation before forming a complete OMR system and studying the process as a whole.

Below we summarise our findings. Like the subdivision of work, we review each stage in turn before concluding the thesis with more general comments.

10.1 Staff detection

Published literature on staff detection focuses on the location of staff lines. Although this is the key task, there are other issues that must not be neglected. Only a few authors have considered works that include different sized staves within the same page, and there is no evidence in the published literature that two staff systems appearing side by side—which often occurs when a musical work includes a coda—are processed correctly. This thesis describes a new approach that addresses both of these complications, and at the same time generalises existing techniques to process staves that consist of an arbitrary number of lines.

In a top down approach, the new algorithm locates potential staff systems, potential staff areas, potential staff segments, and potential staff lines. Consistency checks are then applied to the potential staff lines to determine the true staff lines, and this information is propagated back to eliminate false staves and staff systems. A horizontal methodology forms the basis for the algorithm because it is unaffected by differences in the ver-

tical spacing of horizontal lines, and naturally processes a staff consisting of a single staff line—something which cannot be said for vertical-based methods, as described in existing literature [Car89, Gla89, KI90, Ree95].

Most existing OMR systems are programmed to detect staves consisting of five staff lines. For an extensible system this restriction must be relaxed. However, there is a problem. Because Western text makes prominent use of three distinct heights (the base line, the top of lower case letters, and the top of capitals) we found that text can be mistakenly processed as staves consisting of two or three lines. The contention is eliminated by introducing an OCR preprocessing step that detects and removes all text before staff detection is performed.

A novel aspect in the new staff detection algorithm is the use of the detected staff information to refine the search for staves. Like all other staff detection algorithms, the new algorithm relies upon heuristics to identify staff lines because, at this stage in the OMR process, nothing is known about the size of the music notation. However, heuristics are not guaranteed to be perfect, hence the new algorithm includes a second pass that substitutes the detected staff information in place of the heuristics to obtain more accurate results.

The idea can be taken further. When processing subsequent pages from the same piece of music, the staff information detected on the first page can be used as the starting point for staff detection on other pages.

Through experimentation, tolerances in the new algorithm were empirically chosen to give the best performance. Because the graphical properties of staves in different notations vary considerably, no single set of tolerances successfully processed all stave types. Instead, an OMR system should provide different sets of tolerances, matched to specific notations.

10.1.1 Future work

Future work on this part of the OMR system needs to focus on increased sophistication. For example, the current procedure for identifying the *staff jump*—the minimum distance between two adjacent staves in the same staff system—is simplistic. Error analysis found this step to be the most error prone, highlighting the need for a more sophisticated check. Similar comments apply to the consistency check used to identify the true staff lines.

Savings in computational cost are also possible. For instance, the flood-fill algo-

rithm was chosen for shape extraction because it was straightforward to implement. More efficient algorithms, such as the run-based region filling algorithm [WMB94], exist.

10.2 Musical object location

Much of the complexity in musical object location arises from the superposition of objects on the staff. In attempting to separate the two, a secondary issue becomes avoiding fragmentation. Further complications result from poor notational layout in many works, where individual musical features are drawn so close together that they touch. A practical OMR system must deal with all these complications.

Two popular methods for musical object location are to *remove* the staff lines and to *ignore* the staff lines. To ignore staff lines, emphasis is placed on the shapes that fall between the staff lines. This is undesirable in an extensible system because there is no limit to the musical shapes that can occur. Fortunately, removing staff lines avoids this problem, and is therefore suitable for an extensible OMR system. A less common method for musical object location is based on crossing the staff lines. This too is suitable for an extensible system.

In this thesis we demonstrate that removing and crossing staff lines are equivalent in their abilities. When using a computationally expensive test to detect superimposed musical objects, however, we empirically show the latter method to be significantly faster. For example, using the test devised by Martin and Bellissant [MB91] the removal method takes 2 minutes 35 seconds on average to process an A4 page, compared with 38 seconds by the staff crossing method.

With musical object location so strongly based on the position of staff lines, the accuracy of staff line detection is crucial to the success of the algorithm. The standard technique for staff line removal (or staff line crossing) assumes staff lines are straight, but effects due to aliasing, and deformations introduced through scanning, can distort the location of the staff lines, resulting in disastrous effects (see Figure 4.24 on page 93).

We therefore developed and evaluated two competing strategies that tolerated fluctuations in staff line position. *Wobble* uses the previous slither of staff line to define a generous window in which to locate the next staff line slither. *Track* uses the predicted location of the staff line (modelled as a straight line) as a starting point for a staff line

slither, and then looks above and below this point for a suitable slither if one is not found directly. When tested on the representational cross-section, both strategies had comparable computation costs, but *wobble* was more accurate.

10.2.1 Future work

With current techniques it is inevitable that fragmentation will occur. The difficulty in resolving this issue is strongly related to the locality of the pixels checked. If our view of music was restricted to a moving window a few millimetres wide, we too would have difficulty deciding which sections of staff line belong to superimposed musical features. This is effectively what happens in the computer algorithms described: the decision to remove (or cross) a slither of staff line is based on the configuration of a few surrounding pixels.

The approach in this thesis has been to accept that fragmentation will occur, and to deal with the consequences. This is a reasonable assumption to make because even if it were possible to achieve perfect separation of objects from the staff, other factors such as imperfections in the printing and scanning of music cause fragmentation anyway. Special attention is paid to fragmentation in the primitive detection stage, where mechanisms exist to help process not only fragmented objects, but objects that touch.

Perhaps it is only possible to achieve perfect separation by developing an OMR system modelled more closely on human behaviour, switching backwards and forward between the numerous tasks involved, where necessary. This is an interesting and certainly challenging problem.

10.3 Image enhancement

Image enhancement work focused on two types of improvement: correction for skew, and correction for deformations introduced during scanning.

With staff detection complete, and consequently the angle of skew inferred from the detected staff lines, it is a simple procedure to correct an image for skew. Existing work applies either a rotation operation or a shearing operation to the entire page. Although the shearing operation is typically faster, it only corrects skew in one axis.

This thesis describes a series of improvements to both shearing and rotation based skew correction algorithms:

- Using existing techniques, much of the time spent correcting a page of music is wasted moving "uninteresting" white pixels. By delaying the operation until musical object location is complete, only the located objects and staff lines need correcting, thus pruning away much of the white space. This was empirically shown to have roughly halved the computational cost.

- Because the angle of skew introduced through scanning is typically low—in the corpus, skew rarely rises above 1°—the shearing algorithm can be optimised to shift a byte at a time rather than a bit. Adding this optimisation to the delayed shearing operation was empirically shown to be five times faster.

- Source and destination bitmaps account for the majority of memory usage in rotation and shearing algorithms. This thesis describes how to order the access to pixels in the source bitmap, so as to eliminate the need for the destination bitmap.

Motivated by the need to correct bowing staff lines caused by scanning music from a tightly bound book, a second form of image enhancement investigated is the correction of deformities introduced through scanning. Work focused on deformation in the y-axis.

In principle a staff line is a straight line. Any deviation found in a staff line in the scanned image, therefore, explicitly represents deformation. Based on this observation, a novel algorithm is detailed that attempts to correct deformation in the y-axis.

The deformation captured by a single staff line is local to that part of the image. By adding nearby staff lines into an accumulator, this information is diffused to obtain a more general picture of the deformations occurring in the surrounding area. The accumulator is then processed to generate a discrete mapping from diffused staff line information to a straight, horizontal, line. To cope with non-homogeneous bending, two accumulators are used: one for the top one-third of the image and one for the bottom one-third of the image. The transform for a particular line in the image is generated by interpolating the values in the two accumulators.

10.3.1 Future work

Future work on image enhancement needs to establish the importance of each improvement.

Given that the algorithm for correcting deformation is a generalised form of shearing, and that an OMR system based on a single shearing operation is near identical in accuracy to a rotation-based system (Chapter 9), it is conjectured here that these two steps in the OMR process can be reduced to an application of the correction for deformation algorithm. Experimentation is needed to verify this.

Alternatively, if a thorough survey of skew introduced through scanning verifies that values are as low as the corpus suggests, then it may prove possible to omit all image correction steps. Of course if this scenario was pursued, then pitch calculations must be based on local staff line information. This is simple to arrange if either the *wobble* or *track* procedures for staff detection are used, since they have already traced out the (possibly deviant) staff lines present in the image.

The images in the corpus are generally of a good quality, and therefore no noise filtering work was deemed necessary. If poorer quality works are to be processed reliably, then noise filtering algorithms and general image restoration techniques will become important.

10.4 Primitive detection

The key development this thesis brings to primitive detection work is the specially designed programming language PRIMELA. Designed to abstract the essence of primitive detection work, whilst cutting away much of the tedious, and repetitive detail, the language not only resolves the issue of how an OMR system can process an extensible set of notation, its conciseness trivialises development time. A comparison of the length of primitive descriptions written in a traditional programming language (C++) and PRIMELA showed an average reduction from 1000 lines of code to 60.

Key attributes to the language are:

- an initially neutral strategy towards the processing of primitives that can be customised,
- a supportive collection of pattern recognition routines, and
- the ability to draw arbitrary graphical shapes, scalable to any size without loss of information.

Should the configurer of the system choose to deal with fragmented and touching objects, support is provided by the language. To lessen the impact of fragmentation, two mechanisms exist. The area to be checked can be enlarged, and the original scanned image can be specified as the place to perform the match. To lessen the impact of touching objects, a PRIMELA description can remove primitives as soon as they are detected.

10.4.1 Future work

Future research that refines the work presented here includes:

- a database of geometric information extracted from the detected primitives,
- an enlarged collection of pattern recognition techniques available through PRIMELA,
- optimised pattern recognition techniques,
- improved PRIMELA descriptions,
- an enlarged set of shapes described using PRIMELA, and
- a graphic user interface (GUI) for the development of PRIMELA descriptions.

Now that a sizeable body of primitive shapes has been processed, a database can be built and statistical information gathered. A simple use of the database is to extract information such as the normalised area, perimeter and aspect ratio of a particular primitive type, and to use this to bolster the accuracy of the PRIMELA description without substantially increasing its computational cost. A more significant use of the data is to refine the limits a PRIMELA description uses to pass pattern recognition tests, since the current thresholds were set by trial and error.

A different direction of work to pursue is the use of image compression techniques to reduce the size of the file necessary to store a scanned piece of music. Witten *et al.* describe a compression scheme for scanned images of text, based on a library of isolated shapes [WBE+94] that appears adaptable to musical images. In this method a library of distinct shapes found in the image is constructed. As each shape is extracted from the image, it is matched against the current library and the closest match recorded, along with an error-map of any deviations between the matched library symbol and the current

shape. If no reliable match exists, then the new shape is added to the library. The compressed version of the page consists of the library, the sequence of recorded matches, the x and y offset values between the sequence of extracted shapes, and the error-maps, where standard compression techniques are applied to each section of the new file.

In adapting this scheme for musical images, an obvious alteration is to use a library of primitive shapes rather than a library of isolated shapes. This naturally deals with fixed-size primitives in music notation such as note heads, accidentals, and clefs, but not variable sized primitives such as beams, slurs, and vertical lines.

One solution might be to refine variable sized library symbols to include specialised parameters. For instance, a match with the vertical line symbol in the library would no longer record just the library symbol number, but two extra parameters: width and height; and a beam would require the parameters: thickness, length, and elevation.

10.5 Primitive assembly

Existing work has demonstrated that grammar-based techniques are a viable solution for many stages in the OMR process. Moreover, the structure of a grammar naturally suits the description of an extensible set of notation, making it particularly appropriate for this thesis.

In the past, grammars have been used to solve more than one stage of the OMR process simultaneously, and consequently require sophisticated grammar methodologies. This not only raises the complexity of parser required, but also increases the difficulty in designing grammar rules. By restricting the scope of the grammar, this thesis describes how to accomplish primitive assembly, whilst retaining a simple grammar methodology.

The Definite Clause Grammar (DCG) methodology forms the basis of the work. To deal with the issue of parsing a two-dimensional arrangement of primitives (the tokens of the language) an existing DCG parser (120 lines of Prolog code) was modified to use a bag of primitives rather than a list (65 extra lines of code) and some supporting predicates were added (31 extra lines of code).

A bag is like a set in that the order of objects does not matter, and like a list in that the same element can appear more than once. In terms of run-time complexity, the introduction of the bag data-structure is potentially dangerous since a grammar can now

specify combinatorially large search spaces. Fortunately in this application the configurer is both the specialist in music notation and computer knowledge representation design, and therefore it is easy to guard against such dangers through the prudent use of embedded constraints and the control of backtracking.

Computational time is further reduced by filtering the bag of primitives at key points. Empirical testing showed that a solution which first filters primitives by staff system and then by rules embedded in the grammar performs nearly 60 times faster than an unfiltered approach. When processing an A4 page, for example, the average time was reduced from 30 minutes down to 30 seconds. A caveat to this, though, is that the optimisation is only possible for notations where primitives in the same musical feature lie in close proximity to one another.

A grammar that processed the core notation to CMN was 751 lines long. A grammar that processed the core notation to plainsong notation was 197 lines long. In both cases over half the code written specified constraints and the storing of attributes in the derivation tree.

10.5.1 Future work

Numerous themes can be pursued from this work:

- the existing grammars could be expanded to process a wider range of musical features,
- assembly rules could be added that anticipate primitive detection errors, and
- a more suitable logic programming language could be selected.

Although the assembly work exploits uncertain data to obtain a robust assembly procedure, more work can be done in the area by adding assembly rules that anticipate primitive detection errors. For example, it is common for a bass clef curl to become fragmented, increasing the likelihood of primitive detection failure. To compensate for this, an extra assembly rule for the bass clef could be added that assembles two unclassified shapes and two dots into a bass clef if they meet the requirements of a fragmented bass clef. Alternatively, a vertical stack of note heads is most likely part of a chord. An as-

sembly rule could be added to achieve this, even if the PRIMELA description for a vertical line failed to detect the relevant note stem.

Another source for improvement is the implementation of the parser. Standard Prolog is not necessarily the most suitable language for the BCG parser as there are certain features in the language that are unnecessary for our implementation, particularly dynamic variables. In supporting these features, however, a Prolog compiler introduces complexities that we do not require, and only serve to slow down execution.

A key impediment in the parser is the allowance for dynamic variables. In a language with dynamic variables, the *type* of a variable can change during the execution of a program. While this is a useful paradigm in some situations, it is not one we require for our parser. Some variants of Prolog allow type specification, yielding faster code execution. Modifying the parser to exploit this non-standard extension, therefore, will lower execution time. Alternatively, dynamic compiler techniques exist that detect consistent type usage of a variable, and reconfigure code to take advantage of this [Emb95]. Replacing the current Prolog compiler with one based on such techniques would achieve a similar improvement with less effort.

Further compiler intricacies that encumber execution are uninstantiated variables and back-tracking because many predicates in the parser do not require such generality. The logic programming language Mercury [SHC95] counters both these issues by providing a mode system that controls a predicate's level of determinism. In Mercury the specification of a predicate can range from completely deterministic with only one solution that cannot fail, through to nondeterministic with many solutions where failure is possible. Additionally, Mercury is strongly typed, and therefore naturally encompasses any computational saving gained by re-implementing the parser in a non-standard typed variant of Prolog. In benchmark tests, Mercury is twice as fast as the best Prolog system currently available.

For this thesis, standard Prolog was a convenient choice because a DCG parser implemented in this language was available, and therefore straightforward to modify. It is anticipated that a re-implementation of the parser in Mercury will significantly enhance performance.

10.6 Musical semantics

The final stage for a *complete* OMR system is to derive the musical semantics of a piece. Explanations of this stage are rare in the literature for two reasons: first, many systems do not take the OMR process to its ultimate conclusion, ending instead with the graphical classification of musical features; and second, even when a project does implement this stage, the operations are too specific to the particular implementation to make publication worthwhile.

Broadly speaking, extracting the musical semantics of a work consists of (possibly) multiple passes over a graph-like data structure, creating links, deleting links, and modifying the attributes stored at nodes, due to the effect of one musical feature on other musical features. This thesis shows that existing descriptions fit the model, and goes on to detail a new algorithm that is in keeping with the model and is amenable to an extensible OMR system.

First a basic *lattice-like* graph structure is built. Derivation sub-trees tagged as *musical events* during primitive assembly form the nodes of the graph, and based on *hotspot* information (also stored at the root of each sub-tree) musical events are built into horizontal and vertical *time-threads*. Built on top of horizontal time-threads are *structured lists* that provide different levels of access to musical events: page, staff system, staff, bar, and voice.

Once the basic lattice-like structure is complete, control is passed over to configurer supplied routines. By traversing the structure—modifying nodes and links where appropriate—these routines implement the extraction of the musical semantics for a particular notation.

Compared with other stages in the OMR process, the musical semantics stage is fast. For an implementation aimed at CMN score reconstruction, the average time to process an A4 page to find primitives exceeded 9 minutes, whereas the construction and traversal of the lattice-like graph took less than a second.

10.6.1 Future work

Although the routines implemented to demonstrate the traversal and modification of the lattice-like structure were fast, they were also simplistic and errors occurred (Table 9.9

on page 231). Future work on this part of the OMR system could therefore focus on increased sophistication, without markedly effecting the overall computational cost of the system. For instance, if an accidental does not appear to have a note head at the same pitch adjacent to it, then the pitch of the accidental could be wrong, or the pitch of a nearby note head could be wrong.[1] Algorithms could be encoded to detect such inconsistencies and then try plausible alterations to rectify the anomaly.

A more global analysis of the data adds additional sophistication. In Chapter 9 (page 232) we described a refinement that detects inconsistencies in bar duration and restructures the lattice based on the surrounding information. This idea can be extended further. For example, note pitches and key signatures can be compared for consistent use, and recurring misuses of notation, such as the common practice of dropping the '3' that denotes a triplet after the first few occurrences, can be sought and corrected.

10.7 A complete OMR system

Comparing the accuracy of rival OMR systems has always been difficult: different systems place different restrictions on the type of music processed; different end-points alter what information is extracted from the image; and different file formats suitable for the same end-point impose different structures on how the data is represented. This thesis counters these issues by counting the number of editing operations a hypothetical editor requires to correct the computer generated information into what it should be.

No rigorous definition exists for the hypothetical editor, rather, if we can justify a straightforward editing operation which requires only a few keyboard presses and/or clicks of the mouse, then we assign a cost of '1' to the operation. More complicated operations must be built up from these atomic steps.

The idea of counting edit operations was first suggested by Carter [BC96]. The extension to a hypothetical editor frees us from any idiosyncrasies a particular editor may have. Full score reconstruction was chosen as the end-point because this forms a super-set of the envisaged OMR applications: audio play-back, transposition, and even compression.[2]

[1] A third possibility is that an error has occurred in the primitive detection phase.

[2] An A4 page scanned at 300 dpi and compressed using `gzip` takes around 60,000 bytes, whereas the equivalent OMR generated Lime file takes only 1500 bytes.

The hypothetical editor is integrated with the OMR process, hence there are numerous points at which the editor can correct mistakes. For example, after primitive detection a screen showing all the shapes detected as one particular primitive type can be shown, and the opportunity provided to correct any mistakes. Correcting a missed vertical line on a screen showing all the unclassified shapes, is less work than having to add a new note (possibly involving multiple note heads) into the reconstructed score. We refer to these different opportunities for correction as *correction phases.*

Using this counting measurement for accuracy, two configurations based on robust PRIMELA descriptions were compared. The first configuration corrected an image for skew using a rotation operation. The second configuration used a single shearing operation, and thus skew remained in the vertical axis. Collated results showed that the shearing-based configuration was only marginally faster, but the anticipated rise in errors did not occur, supporting our claim that the devised PRIMELA descriptions were indeed robust.

10.7.1 Future work

The hypothetical editor assigns a cost of '1' to an atomic correction, regardless of the correction phase. This does not necessarily imply that the time taken to locate errors in the different correction phases are the same. A useful future project is a field-study of the time it takes human operators to locate and correct errors in the various phases of the correction process. From this study, an average weighting factor for each correction phase can be estimated.

A more extensive field-study is to survey the types of error made. For instance, it was noticed in the OMR system developed for CMN that music including sections of split voice were more error prone. The imprecise detection of vertical lines in a split voice notation involving both up and down note stems caused errors in primitive assembly, and in other spilt voice situations, errors in the vertical alignment of note heads in the lattice-like structure for musical semantics occurred. By increasing our knowledge of such errors through the survey, modifications can be made to an OMR configuration to counter these problems.

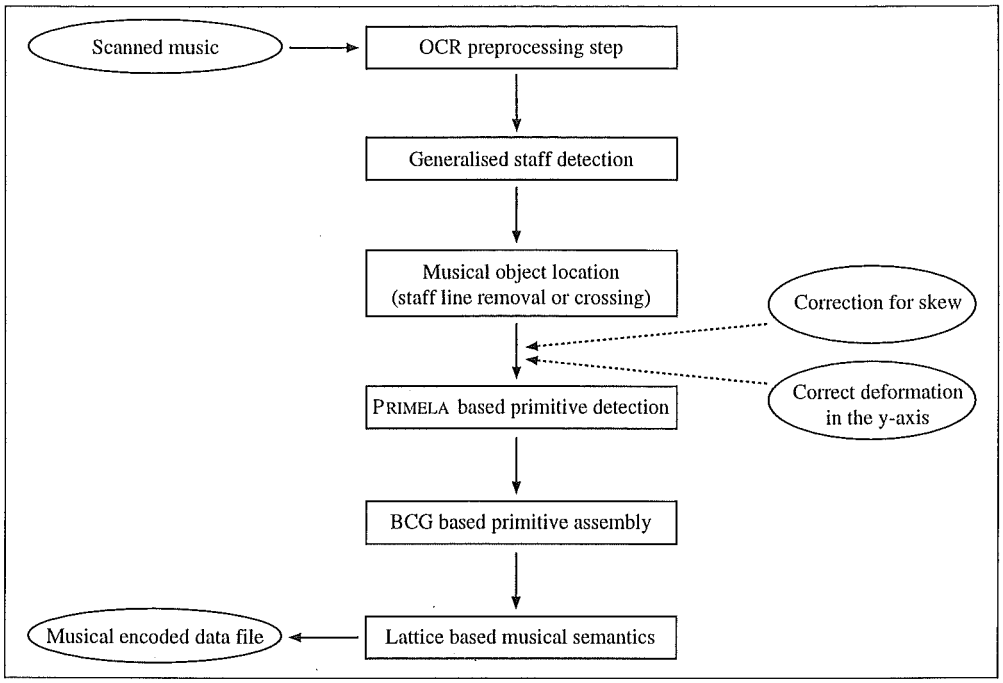


Figure 10.1: The overall structure to the extensible OMR system described in this thesis.

10.8 Conclusion

This thesis does not answer the question, "what is the best way to accomplish OMR?" because the question is too simplistic. The answer depends on the needs of the application.

More significant is how an OMR system is to cope with the near limitless set of symbols that occur, even if we restrict the notation to a single style. This is why the primary aim of this thesis has been to study the requirements of an extensible OMR system. Emphasis was also given to the development of reliable algorithms, rather than computationally efficient algorithms, because future advancements in computer technology will reduce the time an algorithm takes. Within this context, rival algorithms were formulated and compared to establish their strengths and weaknesses.

Figure 10.1 summaries the overall structure to the extensible OMR system described in this thesis. Results confirm this work to be a useful advance in the field of OMR research.

Appendix A

Corpus of music notation

To evaluate the algorithms detailed in this thesis a corpus of music notation consisting of 67 pieces was collected. The corpus is divided into 16 categories based on typographical criteria: orchestrated score, miniature score, accompaniment, monolinear (no chords), popular piano, piano, organ, guitar, violin, hymn, percussion, computer typeset, handwritten, sol-fa, tablature, and plainsong notation. Categories overlap, so one piece of music may fit more than one classification. Table A.1 lists the works that belong to each category.

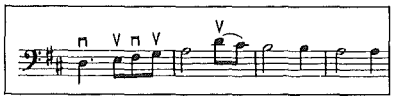
Using a flatbed scanner at a scan resolution of 300 dpi, each piece of music was oriented with care and placed as flat as possible against the scanning window before generating the image. Below we show a characteristic excerpt from each piece along with a brief textual description, where works are listed alphabetically by title. To save space, the excerpts have been reduced in size by 50%.

The only exception to the image acquisition process detailed above is the piece "Bonita" which was obtained from an ftp-site. As a consequence, little is known about the scanning parameters used. The images are copies of the ones used by Martin Roth to test MidiScan. Like the rest of the corpus the scan resolution is 300 dpi.

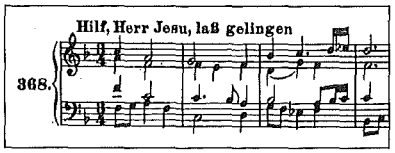
Most entries in the corpus are A4 pages, where one page equates to one scanned image. The main exceptions to this are "Hymns for Today's Church: Tune Index" and "Werke, Reihe II." The format of the former work for each page is a two column list of staves. This was split into two images per page for the corpus. The latter work—an orchestrated score consisting of two staff systems per page—was too large to fit the scanner window, and consequently each page was split in two, based on the staff system boundary.

Category	Titles
Orchestrated score	Qui Belles Amours A, Symphonie D dur, The Cuckoo, Werke, Reihe II
Miniature score	Hymns for Today's Church: Tune Index, L'Autunno, La Primavera, Symphony No. 39 in E Flat
Accompaniment	Bonita, La Provençale, Violin Exercise 1, Violin Exercise 2
Monolinear (no chords)	A Tune a Day for the Violin, Berceuse, Elementary Training for Musicians: Exercise 1, Hymns for Today's Church: Tune Index, La Primavera, Lobe den Herren, Minute Waltz, Music Manuscript Preparation: Exercise 1, Music Reading: Exercise 1, Music Reading: Exercise 2, Music Reading: Exercise 3, Music Reading: Exercise 4, Music Reading: Exercise 5, Shepherds' Song, The Cuckoo
Hymn	Hymns: Ancient & Modern, Hymns for Today's Church: Tune Index, Jesu, Joy of Man's Desiring, Parry My Soul
Percussion	Music Reading: Exercise 3, Music Reading: Exercise 4, Music Reading: Exercise 5
Organ	Fox and Goose, London Bridge, Passacaglia, Prelude and Fugue in G Major, Sleepers, Wake, Ten Little Indians
Guitar	Blues Break 1, Blues Break 2, Introduction et Variations: Sur un Motif de Rossini, Nocturnal, Skittish Gigue, Sonatine
Piano	Bach Chorales: Nos. 366–368, Big My Secret, Once in Royal David's City, Promenade, The Heart Asks Pleasure First
Popular piano	Bohemian Rhapsody, Brothers in Arms, Deck the Halls, Great Balls of Fire, Hazard, Minute Waltz, More Than Words, Pokarekare, Shepherds' Song, The Bonnie Banks o' Loch Lomon', The Hill Street Blues Theme, Wake Up, Little Susie, Ye Banks and Braes
Violin	A Tune a Day for the Violin, Berceuse, Qui Belles Amours A
Computer typeset	Cello Menuet, Duetto uno a Violino e Viola, La Création, Lobe den Herren, Satin Green Shutters (arrangement with chords), Satin Green Shutters (monolinear arrangement)
Handwritten	Elementary Training for Musicians: Exercise 1, Music Manuscript Preparation: Exercise 1
Sol-fa	Jesu, Joy of Man's Desiring, Parry My Soul
Guitar tablature	Blues Break 1, Blues Break 2
Plainsong notation	Cantus ad Libitum, Credo. I, Extra Tempus Paschale, Graduale Sacrosanctæ Romanæ Ecclesiæ In Festis Duplicibus. I, Praefationes in Cantu, The Liber Usualis

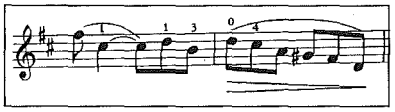
Table A.1: Corpus titles decomposed by category.



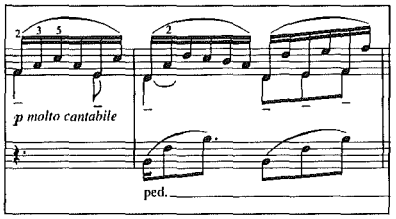
A Tune a Day for the Cello; Categories: cello, and monolinear; Typeset quality: reasonable; Description: includes special markings for cello; Pages: 1; File size: 1049413 bytes.



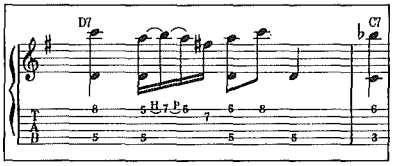
Bach Chorales: Nos. 366–368, by J. S. Bach (Schirmer, 1941); Categories: piano; Typeset quality: poor; Description: includes passages of split voice, and the staff lines on the left-hand side bow (a tightly bound spine prevented the work being pressed flat against the scanner); Pages: 1; File size: 713755 bytes.



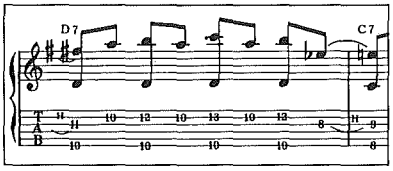
Berceuse, by G. Faurè; Categories: violin, and monolinear; Typeset quality: reasonable; Pages: 1; File size: 1118746 bytes.



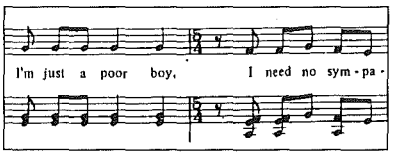
Big My Secret, by M. Nyman (Chester Music, 1993); Categories: piano; Typeset quality: extremely good; Description: includes passages of split voice; Pages: 4; File size: 1052713 bytes.



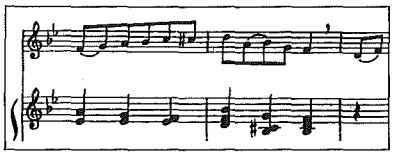
Blues Break 1; Categories: tablature, and chords; Typeset quality: good; Description: includes guitar chords, and special markings for guitar; Pages: 1; File size: 811363 bytes.



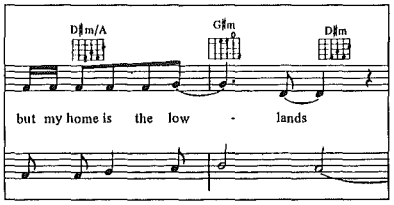
Blues Break 2; Categories: tablature, and guitar; Typeset quality: good; Description: includes guitar chords, and special markings for guitar; Pages: 1; File size: 828022 bytes.



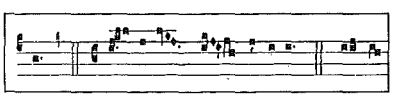
Bohemian Rhapsody, by F. Mercury (Trident Music, 1975); Categories: popular piano; Typeset quality: poor; Description: includes lyrics, guitar chords (written below staff system), passages of split voice, grace sized notes to show second pass variations, marks on original (first page), and inking per page varies; Pages: 9; File size range: 1030336–1062139 bytes.



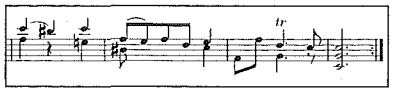
Bonita, by V. Cook (Rubank, 1943); Categories: accompaniment; Typeset quality: reasonable; Description: includes coda inset and on a line by itself, and passages of split voice; Pages: 6; File size range: 929558–1119065 bytes.



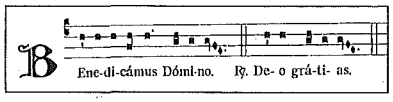
Brothers in Arms, by M. Knopfler (Rondor Music, 1985); Categories: popular piano; Typeset quality: good; Description: includes lyrics, guitar chords (with fingering), two staff systems side by side (a coda), numerous hollow notes, and passages of split voice; Pages: 6; File size range: 989302–1064245 bytes.



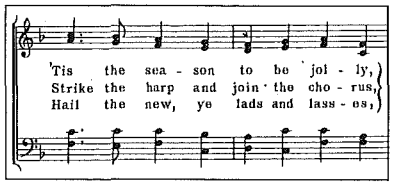
Cantus ad Libitum; Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 1; File size: 347827 bytes.



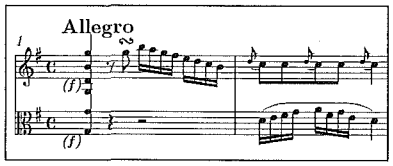
Cello Menuet, by J. S. Bach (Typeset by François Jalbert using MuTEX, 1994); Categories: computer typeset; Typeset quality: reasonable; Description: bitmap generated directly from postscript, and includes passages of split voice; Pages: 1; File size: 357243 bytes.



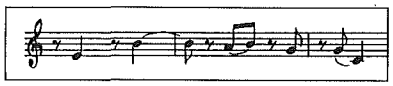
Credo. I; Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 1 ; File size: 424517 bytes.



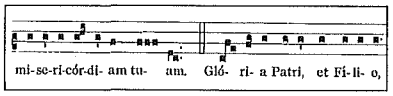
Deck the Halls (traditional Welsh song), by Anonymous; Categories: popular piano; Typeset quality: reasonable; Description: includes lyrics, and noise and pencil marks on original; Pages: 1; File size: 839709 bytes.



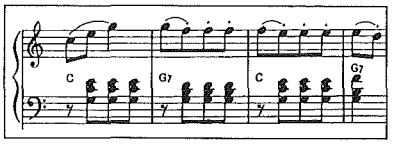
Duetto uno a Violino e Viola, by W. A. Mozart (Werner Icking, 1993); Categories: computer typeset; Typeset quality: reasonable; Description: includes the occasional passage of split voice, and the occasional chords drawn across two adjacent staves; Pages: 11; File size: 1101613 bytes.



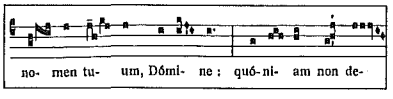
Elementary Training for Musicians: Exercise 1, by P. Hindemith (Associated Music Publishers, 1949); Categories: handwritten, and monolinear; Typeset quality: extremely good (for handwriting); Pages: 3; File size range: 238720–492557 bytes.



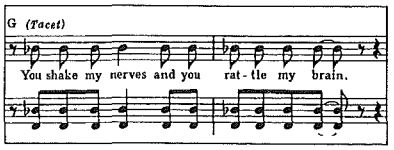
Extra Tempus Paschale; Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 1; File size: 465541 bytes.



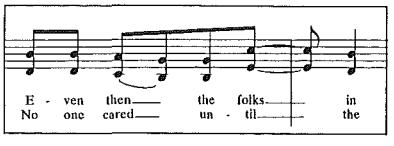
Fox and Goose, by Anonymous (Waseda University, 1985); Categories: organ; Typeset quality: good; Description: includes guitar chords; Pages: 1; File size: 490173 bytes.



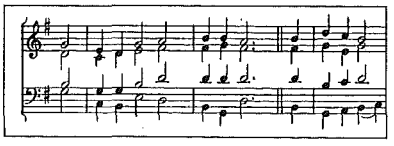
Graduale Sacrosanctæ Romanæ Ecclesiæ de Tempore et de Sanctis (Desclèe & Co., 1974); Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 2; File size range: 363704–382038 bytes.



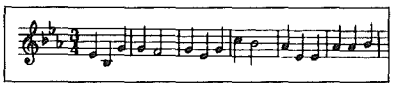
Great Balls of Fire, by J. Hammer and O. Blackwell (BRS Music, 1957); Categories: popular piano; Typeset quality: average; Description: includes lyrics, and guitar chords; Pages: 3; File size range: 948563–964669 bytes.



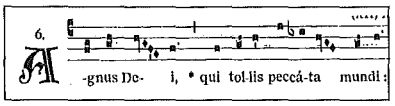
Hazard, by R. Marx (EMI, 1991); Categories: popular piano; Typeset quality: excellent; Description: includes lyrics, and guitar chords (with fingering), two staff systems side by side (a coda), and a section of split voice in one bar; Pages: 4; File size range: 976279–1092343 bytes.



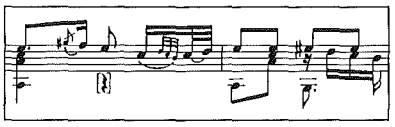
Hymns: Ancient & Modern (William Clowes & Sons, 1950); Categories: hymn; Typeset quality: reasonable; Description: includes passages of split voice; Pages: 2; File size range: 299565–443913 bytes.



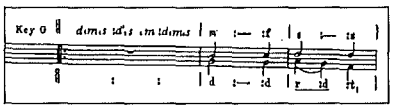
Hymns for Today's Church: Tune Index, by Various; Categories: miniature score, monolinear, and hymn; Typeset quality: reasonable; Description: original is heavily inked; Pages: 36; File size range: 690629–796445 bytes.



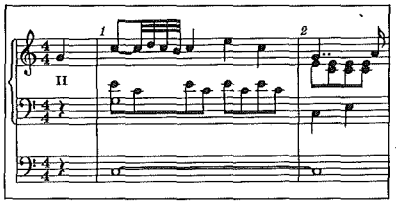
In Festis Duplicibus. I; Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 1; File size: 425953 bytes.



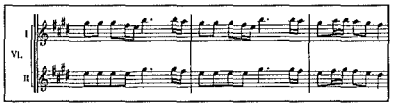
Introduction et Variations: Sur un Motif de Rossini (Editions Chanterelle, 1983); Categories: guitar; Typeset quality: reasonable; Description: includes passages of split voice; Pages: 2; File size range: 970609–988621 bytes.



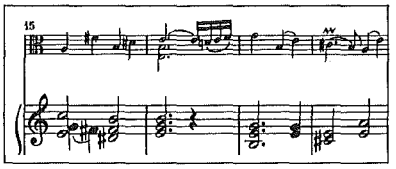
Jesu, Joy of Man's Desiring, by J. S. Bach (Oxford University Press, 1939); Categories: sol-fa, and hymn; Typeset quality: poor; Description: includes marks on original, a high number of hollow note heads, and passages of split voice; Pages: 2; File size range: 497161–508510 bytes.



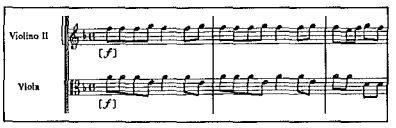
La Création, by J. Haydn (Typeset by Daniel Taupin using MusicTEX, 1990); Categories: computer typeset; Typeset quality: reasonable; Description: includes lyrics, and passages of split voice; Pages: 1; File size: 1052713 bytes.



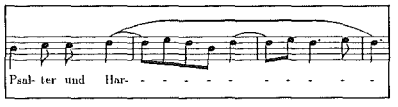
La Primavera, by A. Vivaldi (Ernst Eulenburg, 1982); Categories: miniature score, and monolinear; Typeset quality: reasonable; Description: not all instruments are represented in each staff system; Pages: 5; File size range: 352601–420961 bytes.



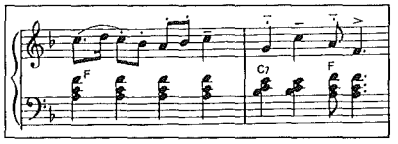
La Provençale, by Louis; Categories: accompaniment; Typeset quality: reasonable; Description: viola piece with piano accompaniment, and includes passages of split voice; Pages: 1; File size: 1077889 bytes.



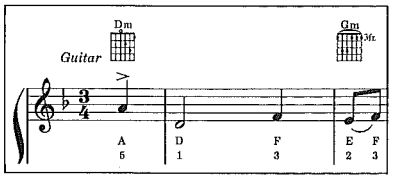
L'Autunno, by A. Vivaldi (Ernst Eulenburg, 1982); Categories: miniature score; Typeset quality: reasonable; Description: includes no slurs or ties, and not all the instruments are represented in each staff system; Pages: 2; File size range: 443731–775669 bytes.



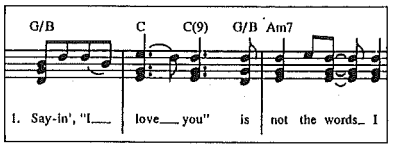
Lobe den Herren, by H. Distler (Typeset by François Jalbert using MuTEX, 1994); Categories: computer typeset, and monolinear; Typeset quality: reasonable; Description: bitmap generated directly from postscript; Pages: 1; File size: 338695 bytes.



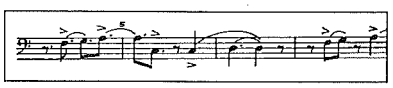
London Bridge, by Anonymous (Waseda University, 1985); Categories: organ; Typeset quality: good; Description: includes guitar chords; Pages: 1; File size: 509309 bytes.



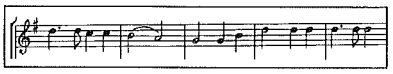
Minute Waltz, by F. Chopin; Categories: popular piano, and monolinear; Typeset quality: extremely good; Description: includes guitar chords (with fingering); Pages: 3; File size range: 552613–817060 bytes.



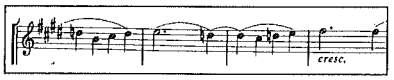
More Than Words, by Bettencourt and Cherone (Funky Publishing, 1990); Categories: popular piano; Typeset quality: reasonable; Description: includes lyrics, guitar chords, grace sized notes to show second pass variations, and passages of split voice; Pages: 4; File size range: 1078258–1096849 bytes.



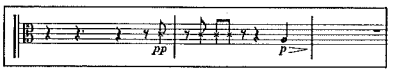
Music Manuscript Preparation: Exercise 1, by M. Mender (The Scarecrow Press, 1991); Categories: handwritten, and monolinear; Typeset quality: good (for handwriting); Pages: 1; File size: 493389 bytes.



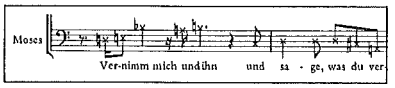
Music Reading: Exercise 1, by V. L. Kliewer (Prentice-Hall, 1973); Categories: monolinear; Typeset quality: reasonable; Description: includes a high number of hollow note heads; Pages: 3; File size range: 497041–506869 bytes.



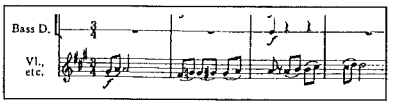
Music Reading: Exercise 2, by V. L. Kliewer (Prentice-Hall, 1973); Categories: monolinear; Typeset quality: reasonable; Pages: 2; File size range: 478473–503923 bytes.



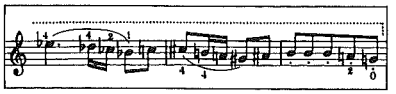
Music Reading: Exercise 3, by V. L. Kliewer (Prentice-Hall, 1973); Categories: percussion (pitched), and monolinear; Typeset quality: reasonable; Description: includes legend at the bottom of the page to explain special notation; Pages: 1; File size: 463357 bytes.



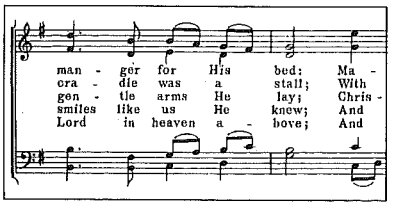
Music Reading: Exercise 4, by V. L. Kliewer (Prentice-Hall, 1973); Categories: percussion (spoken), and monolinear; Typeset quality: reasonable; Description: includes spoken lyrics; Pages: 1; File size: 545237 bytes.



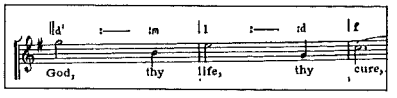
Music Reading: Exercise 5, by V. L. Kliewer (Prentice-Hall, 1973); Categories: percussion (unpitched), and monolinear; Typeset quality: reasonable; Description: includes mixture of 5-line and 1-line staves; Pages: 1; File size: 154430 bytes.



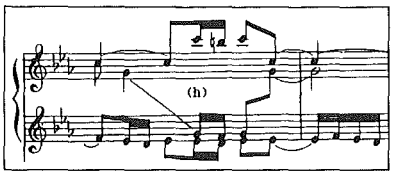
Nocturnal, by B. Britten (Faber Music, 1965); Categories: guitar; Typeset quality: reasonable; Description: includes one section of split voice; Pages: 2; File size range: 960533–963481 bytes.



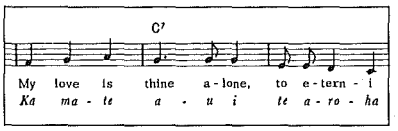
Once in Royal David's City, by C. F. Alexander and H. J. Gauntlett; Categories: piano; Typeset quality: reasonable; Description: includes lyrics, and noise and pencil marks on original; Pages: 1; File size: 837246 bytes.



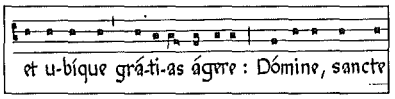
Parry My Soul, by J. S. Bach (Oxford University Press, 1939); Categories: sol-fa, and hymn; Typeset quality: poor; Description: includes marks on original, a high number of hollow note head, and passages of split voice; Pages: 3; File size range: 442973–472693 bytes.



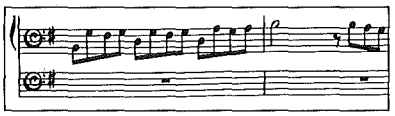
Passacaglia, by J. S. Bach (Novello, 1971); Categories: organ; Typeset quality: reasonable; Description: includes a beamed note crossing staffs within staff system, and passages of split voice; Pages: 1; File size: 1053163 bytes.



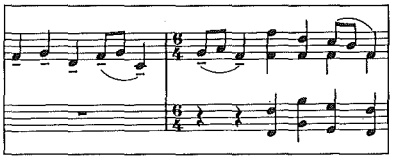
Pokarekare, by P. H. Tomoana (Seven Seas, 1967); Categories: popular piano; Typeset quality: good; Description: includes lyrics, and guitar chords, and passages of split voice; Pages: 3; File size: 1052713 bytes.



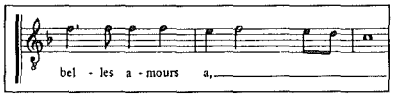
Praefationes in Cantu (L'Abbaye Saint-Pierre, 1971); Categories: plainsong notation; Pages: 2; File size range: 767725–794093 bytes.



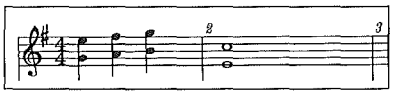
Prelude and Fugue in G Major, by J. S. Bach (Novello, 1971); Categories: organ; Typeset quality: good; Description: warps upwards towards the right-hand side of the first page, includes passages of split voice, and 16th Century style bass clefs; Pages: 2; File size range: 988213–1033733 bytes.



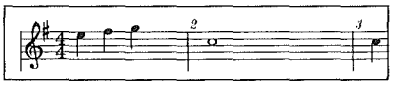
Promenade, by M. P. Mussorgsky (Schott/Universal, 1984); Categories: piano; Typeset quality: extremely good; Description: includes passages of split voice; Pages: 2; File size range: 1033759–1047901 bytes.



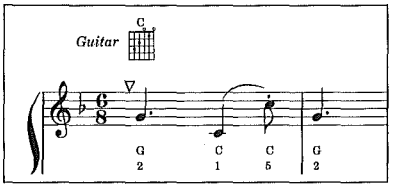
Qui Belles Amours A, by J. des Prez (Oxford University Press, 1975); Categories: score, and monolinear; Typeset quality: reasonable; Description: includes lyrics; Pages: 1; File size: 599854 bytes.



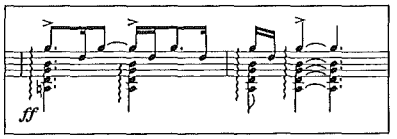
Satin Green Shutters (arrangement with chords), by C. de Burgh (Typeset by David Bainbridge using MusicTEX, 1994); Categories: computer typeset; Typeset quality: good; Description: includes no slurs or ties; Pages: 1; File size: 377949 bytes.



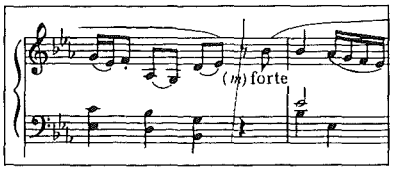
Satin Green Shutters (monolinear arrangement), by C. de Burgh (Typeset by David Bainbridge using MusicTEX, 1991); Categories: computer typeset, and monolinear; Typeset quality: good; Description: includes no slurs or ties; Pages: 1; File size: 552313 bytes.



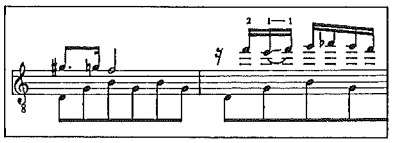
Shepherds' Song, by R. Vaughan Williams; Categories: popular piano, and monolinear; Typeset quality: extremely good; Description: includes guitar chords (with fingering); Pages: 4; File size range: 534327–879073 bytes.



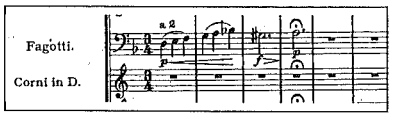
Skittish Gigue, by A. Ritchie (Waiteata Press, 1991); Categories: guitar; Typeset quality: reasonable; Description: first treble clef badly faded in original, and includes passages of split voice; Pages: 2; File size range: 918645–950765 bytes.



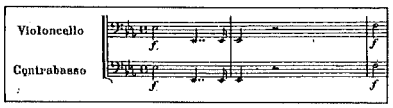
Sleepers, Wake, by J. S. Bach (Oxford University Press, 1941); Categories: organ; Typeset quality: good; Description: includes pencil annotations and markings on original, and passages of split voice; Pages: 5; File size: 1052713 bytes.



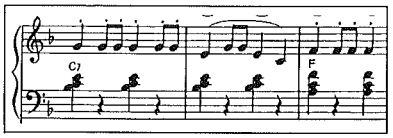
Sonatine, by K. Young (Waiteata Press, 1991); Categories: guitar; Typeset quality: reasonable; Description: includes passages of split voice; Pages: 2; File size range: 893157–909061 bytes.



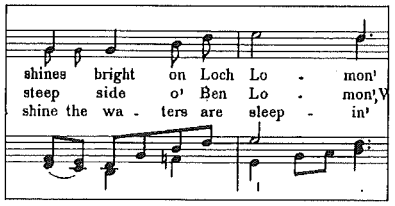
Symphonie D dur, by J. Haydn (C. F. Peters, 1955); Categories: score; Typeset quality: poor; Description: includes passages of split voice; Pages: 2; File size range: 915763–978991 bytes.



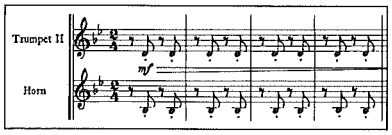
Symphony No. 39 in E Flat, by W. A. Mozart (Boosey & Hawkes, 1941); Categories: miniature score; Typeset quality: poor; Description: includes marks on original; Pages: 1; File size: 392363 bytes.



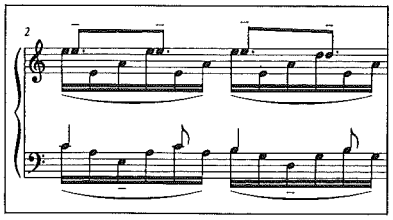
Ten Little Indians, by Anonymous (Waseda University, 1985); Categories: organ; Typeset quality: good; Description: includes guitar chords; Pages: 1; File size: 621517 bytes.



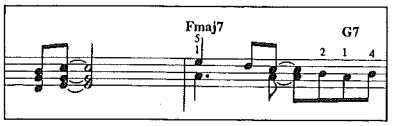
The Bonnie Banks o' Loch Lomon', by H. A. Chambers (Allans Music, 1965); Categories: popular piano; Typeset quality: reasonable; Description: the first staff system merges with black edging in right-hand margin (a tightly bound spine prevented the work being pressed flat against the scanner), and includes lyrics and passages of split voice; Pages: 2; File size range: 537645–1052713 bytes.



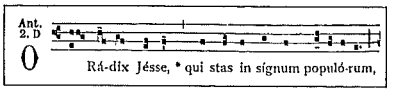
The Cuckoo (traditional Italian/Swiss folk song), by E. Howarth (Chester Music, 1977); Categories: score, and monolinear; Typeset quality: reasonable; Description: include a section of split voice (first page); Pages: 3; File size range: 931435–971245 bytes.



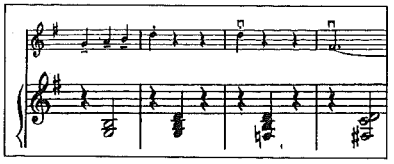
The Heart Asks Pleasure First, by M. Nyman (Chester Music, 1993); Categories: piano; Typeset quality: extremely good; Description: includes passages of split voice; Pages: 6 ; File size: 1052713 bytes.



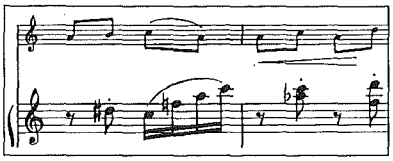
The Hill Street Blues Theme, by M. Post (MTM Music, 1981); Categories: popular piano; Typeset quality: excellent; Description: includes guitar chords, one beamed note crossing from the staff for the left-hand to the staff for the right-hand, a coda inset on a line by itself, and passages of split voice; Pages: 2; File size range: 1052713–1072273 bytes.



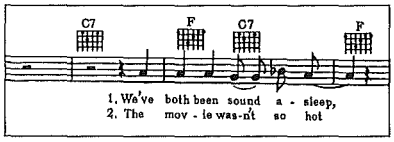
The Liber Usualis (Desclèe & Co., 1963); Categories: plainsong notation; Typeset quality: good (for plainsong notation); Pages: 2; File size range: 302205–316686 bytes.



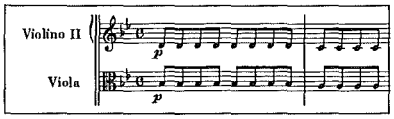
Violin Exercise 1; Categories: accompaniment; Typeset quality: reasonable; Description: violin piece with piano accompaniment (one staff system); Pages: 1 ; File size: 222169 bytes.



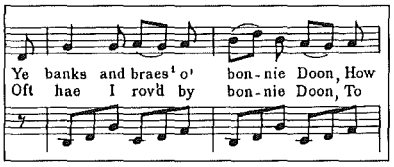
Violin Exercise 2; Categories: accompaniment; Typeset quality: reasonable; Description: violin piece with piano accompaniment (one staff system), and the left-hand piano part uses both treble and bass clef; Pages: 1; File size: 278092 bytes.



Wake Up, Little Susie, by B. Bryant and F. Bryant (Acuff-Rose, 1957); Categories: popular piano; Typeset quality: average; Description: includes lyrics, and guitar chords (with fingering); Pages: 3; File size range: 869869–926272 bytes.



Werke, Reihe II, by J. Haydn (G. Henle Verlag, 1982); Categories: score; Typeset quality: good; Pages: 5; File size range: 564629–684480 bytes.



Ye Banks and Braes, by H. A. Chambers (Allans Music, 1965); Categories: popular piano; Typeset quality: reasonable; Description: includes lyrics, and passages of split voice; Pages: 1; File size: 1052713 bytes.

Appendix B

PRIMELA reference manual

This appendix provides a rudimentary reference manual for PRIMELA. It is not necessary for a reader to be familiar with this material to be able to read the main thesis, but it is here for completeness sake.

The translator/interpreter hybrid model for the language is implemented as a two phase system. First a PRIMELA program is parsed and the portions of the descriptions destined for translation are identified and converted. The resultant code is then automatically compiled. The interpreter is then started. The interpreter oversees the control of the OMR system. In addition to executing the PRIMELA code, its responsibilities include maintaining and processing images, as well as calling the parts of the language that have been pre-compiled. In normal use, only the interpreter is run. The translator need only be invoked if the source code is updated.

B.1 Accessing run-time data

The decision to include an existing programming language as embedded code in PRIMELA greatly simplified its implementation. However, the embedded code must be able to access run-time information from the interpreter, necessitating a communication mechanism between the two. A mechanism based on predefined variables was chosen since it was quick and simple to implement.

Predefined Variable	Description
`rect_xl_`	The x co-ordinate of the left edge of the rectangle selected for searching.
`rect_xr_`	The x co-ordinate of the right edge of the rectangle selected for searching.
`rect_yt_`	The y co-ordinate of the top edge of the rectangle selected for searching.
`rect_yb_`	The y co-ordinate of the bottom edge of the rectangle selected for searching.
`ideal_prim_`	A bitmap version of the graphical shape specified by the current PRIMELA single feature description block.
`copy_prim_`	A bitmap of the shape copied from the image.
`min_staff_infos_`	An array indexed by integers that corresponds to each pixel line in the image. An entry in the array for a given line y, provides information about the *smallest* staff that can place musical features which intersect with that part of the image.
`max_staff_infos_`	An array indexed by integers that corresponds to each pixel line in the image. An entry in the array for a given line y, provides information about the *largest* staff that can place musical features which intersect with that part of the image.
`without_prim_bitmap_`	A bitmap of the scanned image with the staff lines removed. This image is kept up to date with any primitives that are removed.
`slice_matches_`	An array of integers where each entry in the array corresponds to the number of black cross-sections found for each line in the current PRIMELA single feature description block (only in scope when `slice` has been specified).

Table B.1: The predefined variables used by the interpreter that are within the scope of the embedded code for matching control.

B.1.1 Predefined variables

Variables used by the interpreter that are also useful to the control of the matching process are predefined and placed within the scope of all embedded code. These variables are listed in Table B.1. As a convention, all predefined variables end with an underscore (_) so they can be distinguished from regular variables.

B.1.2 User-defined variables

Another area of the interpreter that the embedded code needs to access is the **initialise** block, where user-defined variables are declared. Here we cannot use the predefined variable mechanism directly, since it is not known ahead of time what the variable names will be. The problem is solved by using predefined associative arrays as symbol tables. The interpreter stores a user-defined variable in the associative array appropriate for its type.

Pre-defined Variable	Description
`int_symbol_table_`	Access to the dynamic integer variables in the current PRIMELA single feature description block.
`double_symbol_table_`	Access to the dynamic floating-point variables in the current PRIMELA single feature description block.
`string_symbol_table_`	Access to the dynamic text variables in the current PRIMELA single feature description block.

Table B.2: The predefined variables that allow access to user-defined variables.

Variable name	Type	Values
`option_box`	string	"page" | "object"
`option_pixel`	string	"black" | "white" | "either"
`option_extend_percent`	int	0 to 200
`option_x_extend_percent`	int	0 to 200
`option_y_extend_percent`	int	0 to 200
`option_extend_dir`	string	("u"|"d"|"l"|"r")*
`option_wbitmap`	string	"without prims" | "original page"
`option_filter`	string	"do" | "not"

Table B.3: Permissible values for predefined PRIMELA variables.

This value can then be retrieved in an embedded section of code using the predefined variable name for the symbol table, coupled with the user-defined variable name expressed as a string, so it can be used as the array index. The predefined symbol table names are listed in Table B.2

Figure 6.14 on page 151, therefore, is not quite correct, as it was simplified to aid clarity. Lines 31–32 show user defined variables being accessed directly. The proper definition is given in Figure B.1.

B.1.3 Option variables

The predefined `option` variables, which control where and when the pattern matching occurs, are restricted in the values they can use. The valid values are shown in Table B.3.

B.2 Lexemes

Earlier references to the collective lexical tokens of the PRIMELA language have been informal, relying on an intuitive understanding of terms such as *integer* and *string*. In Table B.4 we define these terms precisely. The remaining lexemes of the language are

```
 1 primitive treble_clef
 2  : size(42,145), origin(21,40), normalise(1.0)
 3 {
 4   yproject ptc
 5     {
 6       transform: scale(0.333333,0.333333), translate(23.333333,106.333333)
 7         {
 8           bspline 2 open 14
 9                     (-70  115) (-01   95) (-02   62) (-61   22) (-48  -39)
10                     (-27 -100) ( 56 -107) ( 43 -133) (-44 -175) (-09 -217)
11                     (-65 -230) (-67 -271) (-29 -293) (-56 -319);
12         }
13     }
14
15   initialise
16     {
17       string option_box = "object";
18
19       int yproject_lower_match = 70;
20       int yproject_upper_match = 75;
21
22       int y_copy_expand  = 2;
23     }
24
25   match
26     {
27       pre       {% %}
28       rect      {% rect_yb_ += y_copy_expand;                          %}
29       copy      {% %}
30       post      {% %}
31       certainty {% linear_certainty(int_symbol_table_("yproject_lower_match''),
                                        score_match_,
32                                      int_symbol_table_("yproject_upper_match''); %}
33       remove    {% true %}
34     }
35 }
```

Figure B.1: The true PRIMELA description for the treble clef.

include *string*
Where *string* is the name of the file enclosed in speech marks.

Figure B.2: The PRIMELA syntax for the include statement.

straightforward keywords. They are defined in the production rules of the language.

B.3 The include statement

Like many other programming languages, PRIMELA has an include statement for importing other files. This is a convenience feature that allows a large program to be split over several files. In PRIMELA a natural decomposition is to use one file per primitive description, and then to gather these files together in a top level file, using the include statement.

Term	Regular Expression Definition
digit	[0–9]
sign	[+|-]
natural	$digit^+$
integer	*sign* $(digit)^+$
float	*sign*? *digit** . $digit^+$
comment	//(.)*\n
identifier	$[\neg(\sqcup\|\backslash t\|\backslash n\|;\|:\|\#\|\backslash\|=\|+\|-\|*\|/\|(\|)\|"\|`\|',)]^+$
string	["][¬("|\n)]*["]

Table B.4: The collective lexemes of PRIMELA defined as regular expressions.

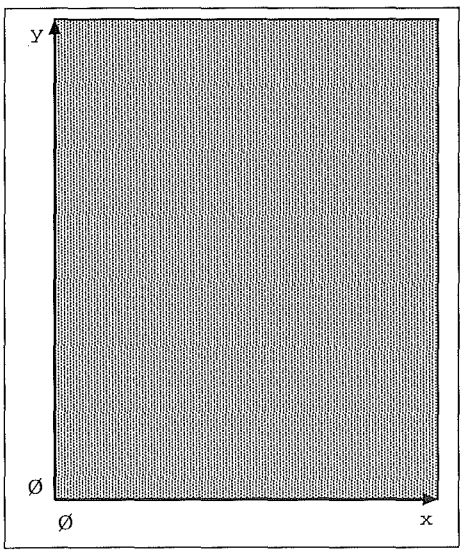


Figure B.3: The co-ordinate system used by PRIMELA for drawing operations.

The syntax for an include statement is given by Figure B.2. From the implementation point of view it is simpler, cleaner and more flexible to implement the construct in the lexical analyser, rather than in the grammar.

B.4 Arbitrary graphics shapes

A co-ordinate system is required for drawing graphical shapes upon a canvas. Here there is a choice in the placement of the origin and the direction of the axes. Unfortunately existing graphic packages do not use a uniform system. In PRIMELA the origin was chosen as the bottom left corner of the canvas, with the x-axis increasing from left to right and the y-axis increasing from bottom to top. This is illustrated in Figure B.3.[1]

[1]This discussion about origins should not be confused with the **origin** construct used at the start of

Matching control keyword	Default behaviour
`pre`	Trivially true.
`rect`	No change in the size of the rectangle.
`copy`	Every pixel in the rectangle is copied.
`post`	Trivially true.
`certainty`	Assigned the value 0.5.
`remove`	No pixels in the rectangle are removed.
`noise`	No filtering takes place.

Table B.5: The default behaviour for the statements that control the execution of a match.

To simplify the task of developing the graphical shapes used by PRIMELA descriptions, a simple graphical editor was developed. The application allows B-spline shapes to be drawn, altered and saved. The resulting file is in PRIMELA syntax, so this can be incorporated into a complete description of a primitive, or re-loaded into the drawing application for further editing. A consequence of designing this drawing application is that the user is shielded from having to know what co-ordinate system is being used. It is hoped that future work will develop a fully integrated graphical system that allows the design of a primitive description, without the user ever needing to know about the underlying PRIMELA language.

B.5 Default behaviour

In PRIMELA, the keywords that control the execution of the matching process are optional (see lines 13–21 in Figure 6.13). If such a statement is omitted in a description, or if the embedded code fragment for the statement is empty, then a default behaviour is automatically substituted. The default behaviours for these statements are listed in Table B.5

a PRIMELA description, which only controls the part of the drawing that is considered the origin when matching. This matching takes place *after* the drawing operations have been carried out.

Bibliography

[AB67] A. Arkadev and E. Braverman, *Teaching computers to recognize patterns*, Academic Press, London, 1967.

[AC82] A. Andronico and A. Ciampa, *On automatic pattern recognition and acquisition of printed music*, Proceedings of the International Computer Music Conference (Venice), 1982, pp. 245–278.

[APFB88] B. Alphonce, B. Pennycook, I. Fujinaga, and N. Boisvert, *Optical music recognition: A progress report*, Proceedings of the Small Computers in the Arts, 1988, pp. 8–12.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principle, techniques, and tools*, Addison-Wesley, Reading, Massachusetts, 1986.

[BAGS92] A. Bulis, R. Almog, M. Gerner, and U. Shimony, *Computerized recognition of hand-written musical notes*, Proceedings of the International Computer Music Conference (San Jose, California), 1992, pp. 110–112.

[Bai91] D. Bainbridge, *Preliminary experiments in musical score recognition*, B.Eng. thesis, Department of Computer Science, University of Edinburgh, Edinburgh, UK, June 1991.

[BB92] D. Blostein and H. S. Baird, *A critical survey of music image analysis*, Structured Document Image Analysis (H. S. Baird, H. Bunke, and K. Yamamoto, eds.), Springer-Verlag, Berlin, 1992, pp. 405–434.

[BBY92] H. S. Baird, H. Bunke, and K. Yamamoto (eds.), *Structured document image analysis*, Springer-Verlag, Berlin, 1992.

[BC90] E. Bernard and D. Cassanet, *Shift invariance and the neocognitron*, Neural Networks (1990), 403–410.

[BC96] D. Bainbridge and N. Carter, *Automatic reading of music notation*, Handbook on Optical Character Recognition and Document Image Analysis (H. Bunke and P. S. P. Wang, eds.), World Scientific, Singapore, 1996, 21 pages (in print).

[BD92] S. Baumann and A. Dengel, *Transforming printed piano music into MIDI*, Advances in Structural and Syntactic Pattern Recognition (Proceedings of International Workshop on Structural and Syntactic Pattern Recognition) (Bern) (H. Bunke, ed.), Series in Machine Perception and Artificial Intelligence, vol. 5, World Scientific, 1992, pp. 363–372.

[BF81] A. Barr and E. A. Feigenbaum, *The handbook of artificial intelligence*, Addison-Wesley, Reading, Massachusetts, 1981.

[Bra90] I. Bratko, *Prolog: Programming for artificial intelligence*, Addison-Wesley, Reading, Massachusetts, 1990.

[BT88] R. Boyle and R. Thomas, *Computer vision: A first course*, Blackwell Scientific, Oxford, 1988.

[Bun82] H. Bunke, *Attributed programmed graph grammars and their application to schematic diagram interpretation*, IEEE Pattern Analysis and Machine Intelligence **4** (1982), no. 6, 574–582.

[Car89] N. P. Carter, *Automatic recognition of printed music in the context of electronic publishing*, Ph.D. thesis, Departments of Physics and Music, University of Surrey, Guildford, UK, February 1989.

[Car92a] N. P. Carter, *A new edition of Walton's Façade using automatic score recognition*, Advances in Structural and Syntactic Pattern Recognition (Proceedings of International Workshop on Structural and Syntactic Pattern Recognition) (Bern) (H. Bunke, ed.), Series in Machine Perception and Artificial Intelligence, vol. 5, World Scientific, 1992, pp. 352–362.

[Car92b] N. P. Carter, *Segmentation and preliminary recognition of madrigals notated in white mensural notation*, Machine Vision and Applications **5** (1992), no. 3, 223–230.

[Car93] N. P. Carter, *A generalized approach to automatic recognition of music scores*, Tech. Report STAN-M-87, Department of Music, Stanford University, California, USA, December 1993, 77 pages.

[Car94a] N. P. Carter, *Conversion of the Haydn symphonies into electronic form using automatic score recognition: a pilot study*, Proceedings of the IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology; Conference 2181 – Document Recognition (San Jose, California) (L. M. Vincent and T. Pavlidis, eds.), February 1994, pp. 279–290.

[Car94b] N. P. Carter, *Music score recognition: Problems and prospects*, Computing in Musicology: An International Directory of Applications **9** (1994), 152–158.

[Cas96] K. R. Castleman, *Digital image processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[CB90] N. P. Carter and R. A. Bacon, *Automatic recognition of music notation*, Proceedings of the International Association for Pattern Recognition Workshop on Syntactic and Structural Pattern Recognition (Murray Hill, New Jersey), June 1990, p. 482.

[CB92] N. P. Carter and R. A. Bacon, *Automatic recognition of printed music*, Structured Document Image Analysis (H. S. Baird, H. Bunke, and K. Yamamoto, eds.), Springer-Verlag, Berlin, 1992, pp. 456–465.

[CBS95] B. Coüasnon, P. Brisset, and I. Stephan, *Using logic programming languages for optical music recognition*, The Third International Conference on the Practical Application of Prolog (Paris), April 1995, pp. 115–134.

[CBT88a] A. T. Clarke, B. M. Brown, and M. P. Thorne, *Inexpensive optical character recognition of music notation: A new alternative for publishers*, Proceedings of the Computers in Music Research Conference (Lancaster, UK), April 1988, pp. 84–87.

[CBT88b] A. T. Clarke, B. M. Brown, and M. P. Thorne, *Using a micro to automate data acquisition in music publishing*, Microprocessing and Microprogramming **24** (1988), 549–554.

[CBT89] A. T. Clarke, B. M. Brown, and M. P. Thorne, *Coping with some really rotten problems in automatic music recognition*, Microprocessing and Microprogramming **27** (1989), 547–550.

[CC94] B. Coüasnon and J. Camillerapp, *Using grammars to segment and recognize music scores*, International Association for Pattern Recognition Workshop on Document Analysis Systems (Kaiserslautern, Germany), October 1994, pp. 15–27.

[CH67] T. M. Cover and P. E. Hart, *Nearest neighbor classification*, IEEE Transactions of Information Theory **13** (1967), 21–27.

[CM87] N. Cercone and G. McCalla (eds.), *The knowledge frontier: Essays in the representation of knowledge*, Springer-Verlag, New York, 1987.

[CPW93] C. H. Chen, L. F. Pau, and P. S. P. Wang (eds.), *Handbook of pattern recognition and computer vision*, World Scientific, Singapore, 1993.

[DK77] R. David and J. J. King, *An overview of production systems*, Machine Intelligence (1977), 300–332.

[EKR90] H. Ehrig, H.-J. Kreowski, and G. Rozenberg (eds.), *Graph grammars and their application to computer science*, Bremen, Germany, March 1990.

[Emb95] H. Emberson, *A dynamic compiler for Scheme*, M.Sc. thesis, Department of Computer Science, University of Canterbury, Christchurch, NZ, 1995.

[FAP89] I. Fujinaga, B. Alphonce, and B. Pennycook, *Issues in the design of an optical music recognition system*, Proceedings of the International Computer Music Conference (Ohio State University), November 1989, pp. 113–116.

[FAPB89] I. Fujinaga, B. Alphonce, B. Pennycook, and N. Boisvert, *Optical recognition of music notation by computer*, Computers in Music Research **1** (1989), 161–164.

[FAPD92] I. Fujinaga, B. Alphonce, B. Pennycook, and G. Diener, *Interactive optical music recognition*, Proceedings of the International Computer Music Conference (San Jose, California), 1992, pp. 117–120.

[FAPH91] I. Fujinaga, B. Alphonce, B. Pennycook, and K. Hogan, *Optical music recognition: Progress report*, Proceedings of the International Computer Music Conference (Montreal, Canada), 1991, pp. 66–73.

[FB91] H. Fahmy and D. Blostein, *A graph grammar for high-level recognition of music notation*, Proceedings of First International Conference on Document Analysis and Recognition (Saint Malo, France), vol. 1, 1991, pp. 70–78.

[FB92] H. Fahmy and D. Blostein, *Graph grammar processing of uncertain data*, Advances in Structural and Syntactic Pattern Recognition (Proceedings of International Workshop on Structural and Syntactic Pattern Recognition) (Bern) (H. Bunke, ed.), Series in Machine Perception and Artificial Intelligence, vol. 5, World Scientific, 1992, pp. 373–382.

[FPA89] I. Fujinaga, B. Pennycook, and B. Alphonce, *Computer recognition of musical notation*, Proceedings of the First International Conference on Music Perception and Cognition, 1989, pp. 87–90.

[FPA91] I. Fujinaga, B. Pennycook, and B. Alphonce, *The optical music recognition project*, Computers in Music Research **3** (1991), 139–142.

[Fuj88] I. Fujinaga, *Optical music recognition using projections*, M.Sc. thesis, McGill University, Montreal, Canada, 1988.

[Fuj92] I. Fujinaga, *An optical music recognition system that learns*, Enabling Technologies for High-Bandwidth Applications (J. Maitan, ed.), vol. SPIE 1785, 1992, pp. 210–217.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics: Principles and practice*, Addison-Wesley, Reading, Massachusetts, 1990.

[Gla89] S. Glass, *Optical music recognition*, B.Sc. thesis, Department of Computer Science, University of Canterbury, Christchurch, NZ, 1989.

[Gro60] D. J. Grout, *A history of Western music*, J. M. Dent and Sons, London, 1960.

[HB93] L. Haken and D. Blostein, *The Tilia music representation: Extensibility, abstraction, and notation contexts for the Lime music editor*, Computer Music Journal **17** (1993), no. 3, 43–58.

[Heu87] G. Heussenstamm, *The Norton manual of music notation*, W. W. Norton, New York, 1987.

[IIHO92] T. Itagaki, M. Isogai, S. Hashimoto, and S. Ohteru, *Automatic recognition of several types of musical notation*, Structured Document Image Analysis (H. S. Baird, H. Bunke, and K. Yamamoto, eds.), Springer-Verlag, Berlin, 1992, pp. 365–376.

[Kas72] M. Kassler, *Optical character recognition of printed music: A review of two dissertations*, Perspectives of New Music **11** (1972), no. 2, 250–254.

[KI90] H. Kato and S. Inokuchi, *A recognition system for printed piano music using musical knowledge and constraints*, Proceedings of the International Association for Pattern Recognition Workshop on Syntactic and Structural Pattern Recognition (Murray Hill, New Jersey), June 1990, pp. 231–248.

[Kim91] F. Kimura, *Handwritten numerical recognition based on multiple algorithms*, Pattern Recognition **24** (1991), no. 10, 969–983.

[LC85] M. W. Lee and J. S. Choi, *The recognition of printed music score and performance using computer vision system*, Journal of the Korean Institute of Electronic Engineers **22** (1985), no. 5, 429–435, In Korean.

[Li91] X. Li, *What's so bad about rule-based programming?*, IEEE Software (1991), 103–105.

[MB91] P. Martin and C. Bellissant, *Low-level analysis and recognition of music drawing images*, Proceedings of First International Conference on Document Analysis (Saint-Malo, France), vol. 1, 1991, pp. 417–425.

[MHS+85] T. Matsushima, T. Harada, I. Sonomoto, K. Kanamori, A. Uesugi, Y. Nimura, S. Hashimoto, and S. Ohteru, *Automated recognition system for musical score – the vision system of WABOT-2*, Bulletin of Science and Engineering Research Laboratory, vol. 112, Waseda University, Tokyo, Japan, September 1985, pp. 25–52.

[Min85] M. Minsky, *A framework for representing knowledge*, Readings in Knowledge Representation (R. J. Brachman and H. J. Levesque, eds.), Morgan Kaufmann, Los Altos, California, 1985.

[MM91] W. F. McGee and P. Merkley, *The optical scanning of medieval music*, Computers and the Humanities **25** (1991), no. 1, 47–53.

[MRHS95] B. R. Modayur, V. Ramesh, R. M. Haralick, and L. G. Shapiro, *MUSER: A prototype musical score recognition system using mathematical morphology*, Tech. report, Intelligent Systems Laboratory, Electrical Engineering Department, University of Washington, Seattle, USA, June 1995.

[MW93] H. R. Myler and A. R. Weeks, *Computer imaging recipes in C*, Prentice Hall, Edglewood Cliffs, New Jersey, 1993.

[Nil82] N. J. Nilsson, *Principles of artificial intelligence*, Springer-Verlag, Berlin, 1982.

[Pae86] A. Paeth, *A fast algorithm for general raster rotation*, Proceedings of Graphics Interface '86 and Vision Interface '86 (Toronto), Canadian Information Processing Society, May 1986, pp. 77–81.

[Pao93] Y.-H. Pao, *Neural net computing for pattern recognition*, Handbook on Pattern Recognition and Computer Vision (C. H. Chen, L. F. Pau, and P. S. P. Wang, eds.), World Scientific, Singapore, 1993, pp. 125–162.

[Pav82] T. Pavlidis, *Algorithms for graphics and image processing*, Computer Science Press, Rockville, Maryland, 1982.

[Pre70] D. S. Prerau, *Computer pattern recognition of standard engraved music notation*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, September 1970.

[Pre71] D. S. Prerau, *Computer pattern recognition of printed music*, Proceedings of the Fall Joint Computer Conference (Montvale, New Jersey), AFIPS Press, November 1971.

[Pre75] D. S. Prerau, *Do-Re-Mi: A program that recognizes music notation*, Computers and the Humanities **9** (1975), no. 1, 25–29.

[Pru66] D. Pruslin, *Automatic recognition of sheet music*, Sc.D. dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, June 1966.

[Rad96] G. M. Rader, *Creating printed music automatically*, Computer (1996), no. 6, 61–68.

[Rea74] G. Read, *Music notation: A manual of modern practice*, Victor Gollancz, London, 1974.

[Ree95] T. Reed, *Optical music recognition*, M.Sc. thesis, Department of Computer Science, University of Calgary, Canada, September 1995.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, MIT Press, Cambridge, Massachusetts, 1986.

[Roa86] C. Roads, *The Tsukuba musical robot*, Computer Music Journal **10** (1986), no. 2, 39–43.

[Ros70] T. Ross, *The art of music engraving and processing: A complete manual reference and text book on preparing music for reproduction and print*, Hansen Press, Miami, 1970.

[Rot94] M. Roth, *An approach to recognition of printed music*, M.Sc. thesis, Eidgenössische Technische Hochschule, Zürich, January 1994.

[RT88] J. W. Roach and J. E. Tatem, *Using domain knowledge in low-level visual processing to interpret handwritten music: An experiment*, Pattern Recognition **21** (1988), no. 1, 33–44.

[Rum90] F. Rumsey, *MIDI systems and control*, Focal Press, London, 1990.

[Rus92] J. C. Russ, *The image processing handbook*, CRC Press, Boca Raton, Florida, 1992.

[San92] A. Sanfeliu, *Syntactic and structural methods in document image analysis*, Structured Document Image Analysis (H. S. Baird, H. Bunke, and K. Yamamoto, eds.), Springer-Verlag, Berlin, 1992, pp. 479–499.

[SER89] C. Stratford, S. M. Embury, and C. J. C. Rowett, *The well tempered scanner*, Honours project, University of Kent, UK, 1989.

[SF94] E. Selfridge-Field, *Optical recognition of music notation: A survey of current work*, Computing in Musicology: An International Directory of Applications **9** (1994), 109–145.

[SHC95] Z. Somogyi, F. Henderson, and T. Conway, *Mercury: An efficient purely declarative logic programming language*, Proceedings of the Australian Computer Science Conference (Glenelg, Australia), February 1995, pp. 499–512.

[Sol89] S. A. Solla, *Learning and generalization in layered networks: The contiguity problem*, Neural Networks: From Models to Applications (Paris) (Personnaz and Dreyfus, eds.), 1989, pp. 168–177.

[Sow91] J. F. Sowa (ed.), *Principles of semantic networks: Explorations in the representation of knowledge*, Morgan Kaufmann, San Mateo, California, 1991.

[Tan87] S. L. Tanimoto, *The elements of artificial intelligence: An introduction using LISP*, Computer Science Press, Rockville, Maryland, 1987.

[WBE+94] I. H. Witten, T. C. Bell, H. Emberson, S. Inglis, and A. Moffat, *Textual image compression: Two-stage lossy/lossless encoding of textual images*, Proceedings of the IEEE; no. 6, June 1994.

[WCAK92] J. Wolman, J. Choi, S. Asgharzadeh, and J. Kahana, *Recognition of handwritten music notation*, Proceedings of the International Computer Music Conference (San Jose, California), 1992, pp. 125–127.

[WMB94] I. Witten, A. Moffat, and T. Bell, *Managing gigabytes: Compressing and indexing documents and images*, Van Nostrand Reinhold, New York, 1994.

[YBD+92] O. Yadid, E. Brutman, L. Dvir, M. Gerner, and U. Shimony, *RAMIT: Neural network for recognition of musical notes*, Proceedings of the International Computer Music Conference (San Jose, California), 1992, pp. 128–131.